Faculdade de Engenharia da Universidade do Porto

Ambiente de desenvolvimento integrado para programação IEC 61131-3

Filipe Rafael dos Santos Ribeiro

VERSÃO DE TRABALHO

Dissertação realizada no âmbito do
Mestrado Integrado em Engenharia Eletrotécnica e de Computadores
Major Automação

Orientador: Mário Jorge Rodrigues de Sousa (Professor Doutor)

25 de janeiro de 2016

Resumo

Esta dissertação documenta o estudo efetuado e o trabalho desenvolvido, incluído num projeto global, que tem como objetivo a criação de um IDE para programação das linguagens da norma IEC 61131-3 em Eclipse. Este projeto encontra-se em desenvolvimento, tendo sido anteriormente realizadas duas dissertações, das quais resultaram o desenvolvimento de um *plug-in* para criação de projetos da norma, um módulo de navegação, um *plug-in* para edição das linguagens textuais, um editor para uma linguagem gráfica e ferramentas para a importação e exportação.

Com objetivo de dar continuidade ao projeto, o trabalho aqui documentado inclui o desenvolvimento de dois plug-ins, um para a criação de variáveis e outro para a criação de tipos de dados, a melhoria do módulo de navegação e a integração deste com os *plug-ins referidos,* bem como a restruturação do código de todo o projeto.

As variáveis e os tipos de dados são elementos comuns definidos pela norma IEC61131-3, que tem como objetivo padronizar a arquitetura, metodologias, elementos e cinco linguagens de programação para os PLC, sendo duas textuais e três gráficas.

Foi também objeto de estudo o Eclipse onde está a ser desenvolvido o projeto. O Eclipse é um IDE, programado essencialmente em Java, que permite o desenvolvimento de novos IDE e *plug-ins,* e é uma plataforma que permite a integração desses e de outros *plug-ins* com o objetivo de expandirem funcionalidades.

Página em branco

Abstract

This thesis documents the study and work of the development done in a global project that aims to develop an IDE for the programming languages of the IEC 61131-3 standard in Eclipse. This project has the work of two theses done previously, which developed a plug-in to create projects of the IEC 61131-3 standard, a Navigator, an editor for the textual languages, an editor for a graphical language and tools to import and export data.

In order to continue the project, it was developed a plug-in to create a variable editor and a plug-in to create an user datatype editor. The Navigator previously created was improved and integrated with these new plug-ins. In addition, the code was restructured in order to improve the project.

The variables and the user data types are common elements defined in the IEC 61131-3 standard, which aims to normalize the architecture, methods and the languages to the PLCs.

This project has been developed in Eclipse, which is an IDE written mostly in Java. Eclipse allows the development of new IDEs and the development of plug-ins capable of extending new functionalities.

Página em branco

Índice

Lista de figuras

Abreviaturas e Símbolos

Lista de abreviaturas (ordenadas por ordem alfabética)

API	Application Programming Interface
EMF	Eclipse Modeling Framework
FBD	Function Block Diagram
FEUP	Faculdade de Engenharia da Universidade do Porto
GEF	Graphical Editing Framework
GMF	Graphical Modeling Framework
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IL	Instruction List
JDT	Java Development Tools
LD	Ladder Diagram
PDE	Plug-in Development Environment
PLC	Programmable Logic Controller
POU	Program Organization Unit
SDK	Software Development Kit
SFC	Sequential Function Chart
ST	Structured Text

Capítulo 1

Introdução

Esta dissertação, realizada no âmbito do Mestrado Integrado de Engenharia Eletrotécnica e de Computadores, documenta o estudo efetuado e o trabalho desenvolvido no projeto de criação de um Ambiente de desenvolvimento integrado para programação das linguagens da norma IEC 61131-3.

Neste capítulo será realizado o enquadramento no qual este projeto se insere (secção 1.1 - Evolução da automação) e explicado em mais pormenor o projeto e detalhados quais os objetivos propostos para esta dissertação (secção 1.2 - Ambiente de desenvolvimento integrado). São depois, apresentadas as motivações para este desenvolvimento (secção 1.3 - Motivação) e, por fim, e exposta a estrutura desta dissertação (secção 1.4 - Estrutura).

1.1 - Evolução da automação industrial

Os últimos tempos apresentaram uma grande evolução da tecnologia e um grande crescimento tecnológico. Pode-se dizer aquilo que na década passada parecia impossível ou impensável, hoje faz parte do dia-a-dia. Analisando essa evolução pode-se acreditar que o que hoje não parece concretizável ou parece mesmo irrealizável poderá fazer parte da rotina na próxima década.

A automação não ficou de parte nesta evolução, tendo sofrido um forte crescimento, resultante também pela exigência da demanda da indústria. A indústria é uma área que contribui para este crescimento da automação, sendo que um dos equipamentos amplamente utilizados na automação industrial é o Controlador Lógico Programável (CLP). Este equipamento é mais conhecido pela sigla inglesa PLC (*Programmable Logic Controller*) e será usada neste documento. Um PLC, na sua forma elementar, é um dispositivo que incorpora um processador, memória, e interface com entradas e saídas, no qual o processador é capaz de executar um programa (série de instruções e operações) gravado na memória que pode ler as entradas e atuar as saídas.

Com os avanços da tecnologia, estes dispositivos têm sofrido uma grande evolução, que tem permitido que sejam cada vez mais pequenos, mais robustos e que tenham mais funcionalidades. Com este desenvolvimento o PLC sofreu uma grande procura, que promoveu um forte crescimento e diversificação: ao nível do equipamento físico em si (com novas tecnologias e com novas funcionalidades); ao nível de crescimento da oferta de dispositivos, com novas empresas; também ao nível das suas linguagens de programação criadas pelas respetivas empresas [1].

Cada empresa desenvolve os seus dispositivos, com características próprias, com as suas linguagens próprias e ambientes de programação próprios, o que faz com que este sector adquira uma grande diversidade. Esta diversidade, do ponto de vista da utilização deste dispositivo, faz com que o utilizador tenha de ter uma maior e constante formação. Além da necessidade de grande conhecimento, esta diversidade apresenta questões de compatibilidade entre diversos dispositivos. O que por vezes implicaria que, com o objetivo de atualizar um equipamento ou com o objetivo de inserir um novo equipamento necessário, seria necessário atualizar todos os equipamentos relacionados com este.

Com o objetivo de padronizar os dispositivos e seus interfaces, as suas linguagens e metodologias de programação, as suas comunicações, definir terminologias, entre outros elementos relacionados com este tipo de dispositivos, a Comissão Eletrotécnica Internacional (IEC, do inglês *International Electrotechnical Commission*) criou e publicou a norma IEC 61131 [1]. Esta publicação, como era seu objetivo, padronizou a área dos PLC, permitindo assim que dispositivos de diferentes empresas possam comunicar e se interligar, e também que se possam interligar equipamentos acessórios de diferentes empresas, tendo a vantagem de usar sempre as mesmas linguagens.

Essas linguagens são definidas na terceira parte da norma, IEC 61131-3. São especificadas cinco linguagens: duas textuais, “Lista de Instruções” (IL) e “Texto Estruturado” (ST), e três gráficas “Diagramas de Ladder” (LD), “Diagrama de Blocos Funcionais” (FBD) e a linguagem “Diagrama Sequencial de Funções” (SFC).

No entanto, esta norma não define um conceito e regras para a criação de um ambiente de desenvolvimento integrado (IDE, do inglês *Integrated Development Environment*) onde seriam desenvolvidas essas linguagens, não define como é guardado o código produzido nessas linguagens, nem define ferramentas de partilha desse código.

Essa falta de definição proporciona, que as empresas no desenvolvimento dos seus IDE proprietários, criem a sua própria forma de gravação de informação (tipo de ficheiros, estrutura, …), a sua própria apresentação e interação gráfica (por vezes muito diferentes de IDE para IDE) e não incluem ferramentas de partilha dos projetos, que são desenvolvidos para os PLC, entre diferentes IDE.

Esta situação apresenta dois novos problemas, o problema de adaptação a diferentes IDE e, apesar de os dispositivos serem padronizados e compatíveis, o problema de quando se pretender alterar o dispositivo, poder ser necessário desenvolver todo o projeto novamente.

1.2 - Ambiente de desenvolvimento integrado para programação IEC 61131-3

Para que o desenvolvimento deste novo IDE seja uma possível solução para os problemas referidos, não pode ser apenas mais um IDE fechado sem possibilidade de integração de outros desenvolvimentos, e sem a criação de ferramentas que permitam a troca livre de projetos já desenvolvidos.

"*The Eclipse Project provides a kind of universal tool platform - an open extensible IDE for anything and yet nothing in particular*" [3], esta é frase inicial de definição de missão do Eclipse e pode ser traduzida como "O Projeto Eclipse fornece um tipo de plataforma universal de desenvolvimento – um IDE aberto e extensível para tudo e para nada em particular". Com o objetivo de permitir a integração de outros desenvolvimentos, este projeto será desenvolvido em Eclipse que permite a criação de extensões que facilmente podem estender as funcionalidades do IDE criado. Esta extensão permitirá que as empresas, que desenvolvem equipamentos, possam criar extensões que acrescentem funcionalidades ao IDE, apresentando assim, uma mais-valia para a empresa.

Para que o IDE permita a partilha de projetos desenvolvidos, este será implementado com ferramentas que exportem o projeto para ficheiro de texto e que possibilitam exportar para e importar de um ficheiro XML (*eXtensible Markup Language*), que seguirá a interface PLCOpen XML, definida pela TC6-XML (TC, Comissão Técnica).

Como referido na introdução do capítulo, a presente dissertação apresenta o desenvolvimento de uma parte do projeto do IDE. Este projeto tem como objetivo a criação de um IDE para as linguagens de programação da terceira parte da norma ***IEC61131***, baseado no programa Eclipse.

Este projeto foi previamente iniciado tendo havido já o desenvolvimento de duas dissertações. A primeira dissertação foi realizada por Filipe Ramos[4] que iniciou o projeto com a criação dos editores para as linguagens textuais (IL e ST), com a criação de um editor para a linguagem SFC e com a criação de ferramentas para a conversão dos documentos das referidas linguagens para a linguagem XML e para a conversão da linguagem SFC para linguagem textual. A segunda dissertação, que foi realizada por José Ferreira [5], contribuiu com a criação de uma ferramenta para interpretação de ficheiros em linguagem XML para os editores existentes, com o desenvolvimento de dois *plug-ins*[1], um para a criação de um projeto e outro para a criação de uma janela de navegação da estrutura dos projetos.

Assim, o desenvolvimento desta dissertação apresenta três partes. Na primeira parte será feito o reconhecimento do que já foi realizado, de forma a compreender a linha de desenvolvimento e averiguar o que poderá ser reutilizado. A segunda parte consiste no desenvolvimento dos objetivos propostos nesta dissertação. A terceira parte será o desenvolvimento dessas ferramentas sustentadas numa possível evolução futura e planeamento do que se poderá vir a ser realizado no seguimento deste projeto.

Os objetivos propostos para esta dissertação são:

1) Implementação de um *plug-in* para a criação de variáveis.

[1] *Plug-in* pode ser traduzido para português como "módulo de expansão", no entanto, nesta dissertação será mencionado como *plug-in*, uma vez que é este o termo usado para os módulos de expansão do Eclipse, mesmo na comunidade de língua portuguesa.

2) Melhorar o *plug-in* que apresenta a árvore de navegação dos projetos da norma IEC61131-3.
3) Implementação de um *plug-in* para a criação de novos tipos de dado.
4) Restruturação do projeto global e dos códigos dos *plug-ins* (objetivo acrescentado no decorrer da dissertação).
5) Implementação da ferramenta para fazer a importação de um projeto a partir de um ficheiro XML e a exportação de um projeto para ficheiro XML ou TXT.

1.3 - Motivação

Este projeto é um desafio muito interessante. Apresenta diferentes oportunidades que ajudarão a desenvolver o conhecimento, nomeadamente da norma IEC61131-3, que terá de ser devidamente estudada para ser possível realizar o IDE seguindo rigorosamente as regras da norma; e também da linguagem Java, uma vez que o desenvolvimento do IDE será realizado nessa linguagem.

Permite obter uma forte experiência em estudo de programa previamente realizado por outros e na restruturação de código para poder ser reutilizado. O facto de ter de se estudar o código já efetuado, para possível utilização desse código para adaptação dos novos *plug-ins* ao projeto existente. Devido, também, ao desenvolvimento de novo código deverá ser feito com o objetivo de poder ser reutilizado e integrado em novas funcionalidades.

Também permite obter uma forte experiência e conhecimento na plataforma Eclipse e nas suas ferramentas. Conhecer a sua arquitetura, a sua interação e um melhor conhecimento com os *plug-ins* existentes e o seu desenvolvimento. É uma forte motivação uma vez que o Eclipse é uma plataforma muito usada e que está em crescimento devido a esta sua potencialidade.

Por fim, existe também a forte motivação de poder fazer parte de um projeto da criação de um IDE que poderá vir a ser útil e utilizado no futuro profissional, solucionando um problema conhecido e sentido.

1.4 - Estrutura da dissertação

De modo a facilitar a leitura e a perceção desta dissertação, esta foi escrita de forma a que o leitor seja apresentado ao tema, percebendo o enquadramento e objetivo, depois possa ter conhecimento do estudo que foi realizado e quais os IDE que atualmente existem, sendo depois descrito o desenvolvimento do projeto e apresentados e discutidos os resultados desse mesmo desenvolvimento. Deste modo a dissertação foi dividida em 5 capítulos, nos quais, no início é realizada uma pequena apresentação das suas secções: Capítulo 1 - Introdução, este é o presente capítulo onde é feita uma apresentação desta dissertação; Capítulo 2 - Estado de arte, neste capítulo é apresentado o estado da arte, é apresentado com detalhe a norma e o funcionamento do Eclipse, assim, como outros IDE existentes; Capítulo 3 Desenvolvimento, neste capítulo é detalhado o trabalho realizado para esta dissertação; Capítulo 4 - Testes, após o desenvolvimento, são expostos os resultados desse desenvolvimento; Capítulo 5 - Conclusões, onde são apresentadas as conclusões do trabalho realizado.

Capítulo 2

Estado de arte

Neste capítulo será exposto o estado da arte relativamente ao tema do projeto. Começará por apresentar os temas relacionados com o projeto com ordem de importância, depois serão apresentados os produtos semelhantes que existem no mercado, por fim será apresentado o desenvolvimento atual do projeto onde esta dissertação se insere.

Assim começará por ser apresentada a norma IEC 61131 (secção 2.1 – Norma IEC 61131), sendo realçada a parte três, que é a parte que se foca o desenvolvimento da dissertação. Na segunda secção será apresentado o Eclipse (secção 2.2 - Eclipse), aqui será explicado o Eclipse, a sua arquitetura e os seus *plug-ins*. Na secção seguinte serão identificados alguns IDE existentes (secção 2.4 – IDE existentes para a norma IEC61131-3), onde serão apresentados.

2.1 - Norma IEC 61131-3

A norma IEC 61131 foi publicada pela Comissão Eletrotécnica Internacional (IEC) em 1992 e é constituída atualmente por nove partes. Esta norma tem como objetivo padronizar os PLC e seus periféricos associados, os requisitos e testes dos equipamentos, as suas linguagens de programação, as suas comunicações, definir terminologias, entre outros elementos relacionados com este tipo de dispositivos [6].

A terceira parte da norma IEC 61131, IEC 61131-3, que foi primeiramente publicada no ano de 1993, é a parte que será estudada e aprofundada neste documento. Este estudo é baseado na segunda edição que foi publicada em 2003. No entanto, existe uma versão mais recente publicada em 2013.

A norma IEC 61131-3 é normalmente referida como a parte da norma que define as linguagens de programação dos PLC. São especificadas nesta norma cinco linguagens de programação: duas linguagens textuais, IL e ST, e três linguagens gráficas, LD e FBD e a SFC. Porém a definição da norma vai além da definição das linguagens.

Para melhor se perceber a norma pode-se separar as suas definições em dois grupos, os elementos comuns e as linguagens, ver Figura 2.1 [7]. O primeiro grupo inclui todos os elementos que são à definição e organização de um projeto e todos os elementos que serão comuns a todas as linguagens. Esses elementos são Variáveis, Tipo de Dados, POU,

Configuração, Recursos e Tarefas. O segundo grupo inclui a definição das linguagens de programação. Nas subsecções seguintes é detalhado em maior pormenor estes elementos e linguagens.

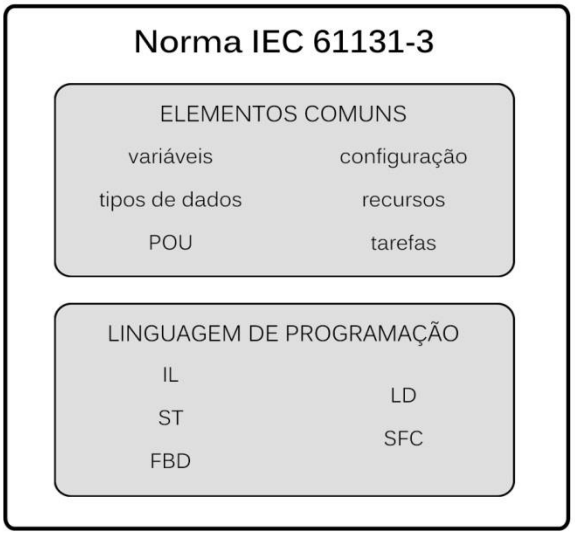

Figura 2.1 - Separação da norma em dois grupos [8].

2.1.1 - Lista de Instruções (IL)

A linguagem IL é uma linguagem textual. Das cinco linguagens da norma é a linguagem considerada de baixo nível, porque ao nível de definição é uma linguagem que se assemelha à linguagem máquina, por exemplo como a linguagem "Assembly". Apesar de assim ser considerada, esta linguagem permite abstração e o uso de POU criados nesta ou em outras linguagens.

Esta linguagem é utilizada para operações mais simples, uma vez que trabalha com operações mais ao nível da máquina, o que permite uma melhor perceção das operações e tempos usados ao nível do processador. Esta linguagem pode ser usada para funções mais complexas, porém apresenta uma maior dificuldade de interpretação quando comparada com as restantes linguagens, uma vez que envolve a criação de um maior número de linhas de código.

```
Network 1: Begin Sinchronizing communications

INI:  AN    "Plc_Start_Delay"          M255.3          -- Delay for entering CPU Run Mode
      L     S5T#4S
      SD    T    126
      AN    T    126
      JC    INI

Network 2: Title:
      A     T    126
      =     "Plc_Start_Delay"          M255.3          -- Delay for entering CPU Run Mode

Network 3: Title:
      ON    "OFF"                      M255.0          -- Always OFF bit
      O     "OFF"                      M255.0          -- Always OFF bit
      S     "1_Cycle"                  M255.2          -- First Program Cycle

Network 4: Title:
      SET
      S     M    310.0
      S     M    320.0
```

Figura 2.2 – Exemplo de linguagem de programação IL.

2.1.2 - Texto Estruturado (ST)

A linguagem ST é uma linguagem textual que se assemelha à linguagem "C". Esta é uma linguagem considerada de alto nível, sendo mais complexa e permitindo um maior detalhe. Apesar disso, é de fácil assimilação para desenvolvimento de cálculos e operações com múltiplos testes e contactos, pois torna-se uma programação mais limpa e de rápida visualização e perceção do código.

Uma das vantagens desta linguagem é a possibilidade de criação de blocos de código de iteração, de condicionalismo e de repetição, devido à existência de comandos básicos usados nas linguagens de programação de alto nível, como por exemplo, os comandos "if", "case", "while", "repeat", "for", … Este facto torna-a adequada para desenvolvimentos mais complexos e repetitivos.

```
(*
------------
Blackbox_FB
------------

Triggers a blackbox.
*)

case State_BlackBox of
    1:  if NDC8.Blackbox.TrigReason = '' then
            NDC8.Blackbox.TrigReason := 'PLC';
        end_if;
        NDC8.BlackBox.TrigEN := True;
        Finished := False;
        State_BlackBox := 2;

    2:  if not NDC8.BlackBox.TrigEN then
            State_BlackBox := 1;
        elsif NDC8.BlackBox.TrigENO then
            NDC8.BlackBox.TrigEN := False;
            State_BlackBox := 3;
        end_if;

    3:  if not NDC8.BlackBox.TrigENO  then
            State_BlackBox := 1;
            Finished := True;
        end_if;
end_case;
```

Figura 2.3 – Exemplo de linguagem de programação ST.

2.1.3 - Diagrama de Ladder (LD)

A linguagem LD é uma linguagem gráfica. Foi a primeira linguagem utilizada para a programação dos PLC, tendo sido criada com o objetivo de ser utilizada por técnicos e engenheiros eletricistas, pelo que a sua visualização gráfica se assemelha um esquema elétrico de contactos e atuadores (sendo também é conhecida por diagrama de contactos).

Graficamente apresenta do lado esquerdo uma barra vertical que corresponde à linha de potência e apresenta uma barra vertical à direita que representa a linha de massa. A ligar as duas barras apresentam-se contactos que, dependendo do tipo e estado atual, vão permitindo ou impedindo o "fluxo da corrente" da barra de potência até à massa, forçando o atuador a ficar ativo ou desativo, dependendo do seu tipo. Com o desenvolvimento da linguagem, esta começou a apresentar não só contactos, mas também funções e blocos funcionais.

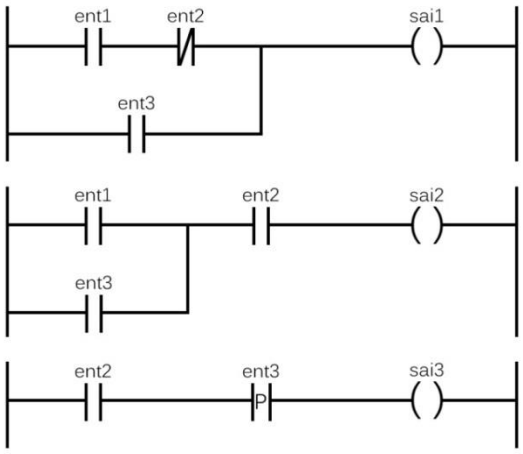

Figura 2.4 – Exemplo de linguagem de programação LD.

2.1.4 - Diagrama de Blocos Funcionais (FBD)

A linguagem FBD é também uma linguagem gráfica. À semelhança da LD é uma linguagem que se tenta aproximar de um esquema elétrico, porém o conceito da FBD é a interligação de blocos funcionais.

A visualização gráfica apresenta a interligação entre os blocos, com funções definidas e compostos por entradas e saídas, permitindo de forma simples e visual compreender quais as condições e funções que influenciam uma determinada saída ou ação. A interligação dos blocos é realizada por linhas que conectam as entradas e saídas. Essas ligações são orientadas transmitindo a informação da esquerda para a direita, podendo existir a multiplicação de uma linha em várias para distribuir a informação por diversos blocos.

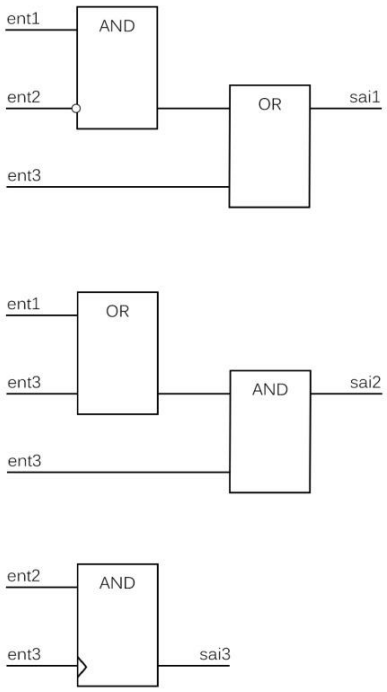

Figura 2.5 – Exemplo de linguagem de programação FBD.

2.1.5 - Diagrama Sequencial de Funções (SFC)

A linguagem SFC é uma linguagem com uma visualização gráfica que permite também um desenvolvimento textual. É baseada na metodologia Grafcet (norma IEC 848), uma metodologia francesa que tem a sua fundamentação matemática nas Redes de Petri.

Esta linguagem é uma sequência de passos e transições alternados e de existência de tokens que percorrem e ativam esses passos mediante a transição seguinte permitir essa passagem. Os passos podem realizar ações, que podem ser, por exemplo, ativar saídas ou chamar um POU, e as transições representam condições, que podem ser, por exemplo, uma entrada estar ativa ou um conjunto uma entrada estar ativa e o resultado de um POU estar desativo. Um token é iniciado num passo específico, denominado por passo inicial, pelo que este passo está inicialmente ativo, quando a transição seguinte a esse passo está ativa o token passa ao passo seguinte, ficando este ativo, e o primeiro passo desativo, e assim sucessivamente.

Além destes três elementos básicos (passos, transições e tokens) esta linguagem tem elementos que possibilitam um controlo de passos em paralelo, em que quando uma transição fica ativa, o token pode multiplicar-se para dois passos seguintes existentes em paralelo. Enquanto as anteriores linguagens apresentam apenas um controlo do tipo sequencial da sua programação, a SFC permite assim um controlo em paralelo.

Esta linguagem apresenta assim, uma grande vantagem face às restantes linguagens, porém as suas ações e operações são elementares, pelo que esta linguagem está normalmente associada às restantes linguagens.

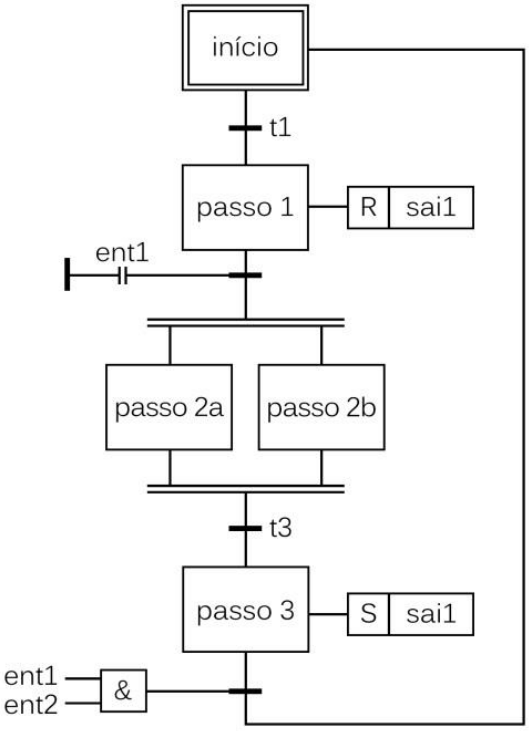

Figura 2.6 – Exemplo de linguagem de programação SFC.

2.2 - Eclipse

O Eclipse começou por ser uma tecnologia da *Object Technology International*, uma empresa subsidiária da IBM. Em Novembro de 2001, foi anunciado o projeto de código aberto Eclipse por um consórcio formado por nove empresas, do qual fazia parte a IBM e ao qual doou o projeto inicial. Atualmente o Eclipse Project pertencente à Eclipse Foundation, uma corporação sem fins lucrativos, que foi fundada em 2004.

O Eclipse é uma plataforma de *software* que foi criada para ser expansível através de *plug-ins*. Além da plataforma base o Eclipse Project possui outros projetos, que são desenvolvimentos de *plug-ins* que têm como objetivo desenvolver, promover e expandir a sua plataforma e os novos *plug-ins*.

"The Eclipse Project provides a kind of universal tool platform - an open extensible IDE for anything and yet nothing in particular."[3]

A frase anterior pode ser traduzida como: "O Projeto Eclipse fornece um tipo de plataforma universal de desenvolvimento – um IDE aberto e extensível para tudo e para nada em particular", esta é a frase inicial da descrição da missão do projeto Eclipse.

O Eclipse surgiu com o objetivo de criar uma plataforma de elevada qualidade que permitisse uma fácil e alargada integração de diversas ferramentas de desenvolvimento de programas. Pretendia-se que o programador pudesse, numa só plataforma de desenvolvimento, ter o máximo de ferramentas disponíveis para as mais variadas linguagens de programação. De forma a promover as potencialidades do projeto e para que a comunidade tivesse conhecimento e pudesse experimentar e desenvolver *plug-ins* no e para o Eclipse, foi desenvolvido o Eclipse JDT, um dos mais avançados e completos IDE de programação Java, e o Eclipse PDE, ambiente de desenvolvimento de *plug-ins*. A Figura 2.7 ilustra a arquitetura do Eclipse.

A simples palavra "Eclipse", referenciada a este projeto pode ter diferentes interpretações, pode ser interpretada como todo o projeto (Eclipse Project), pode ser interpretada como o subprojecto da plataforma (Eclipse Platform) ou pode ser interpretada como um IDE para Java, o ambiente de desenvolvimento integrado do Eclipse com as suas ferramentas de desenvolvimento Java (Eclipse JDT - Java Development Tools). O motivo desta múltipla interpretação advém do fator de diferenciação do Eclipse quando comparado com a concorrência ser a sua plataforma, advém do facto de o Eclipse ter ganho muita popularidade com o uso do IDE de Java, e da sua verdadeira definição que é um conjunto de subprojetos. No entanto, nesta dissertação a definição de Eclipse atribui-se ao "Eclipse Project".

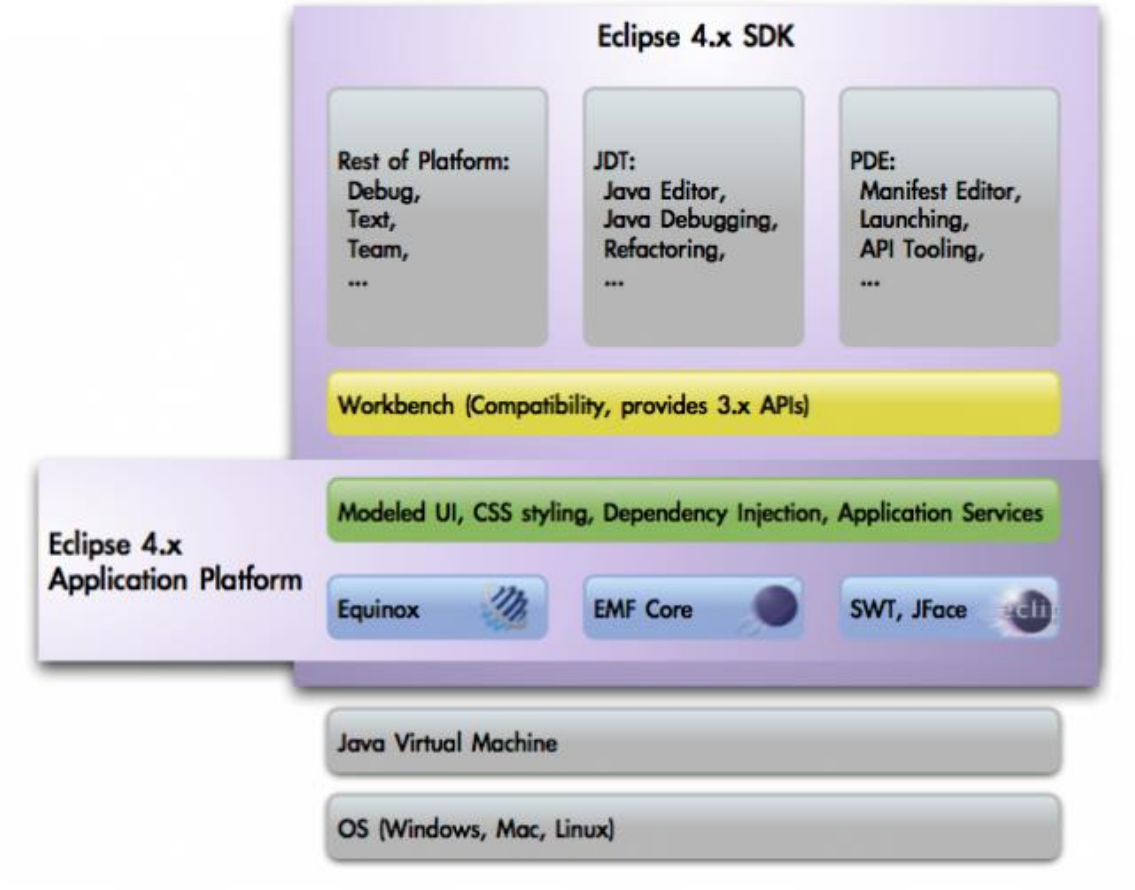

Figura 2.7 - Arquitetura do Eclipse SDK.

2.2.1 - Plataforma Eclipse

A plataforma Eclipse é a base do objetivo do Eclipse, pelo que o desenvolvimento desta plataforma foi complexo e meticuloso para que a plataforma conseguisse responder aos requisitos e necessidades da comunidade.

Uma das principais vantagens da Plataforma Eclipse é o conceito existente, que potencia uma extensibilidade do programa base com outros programas e ferramentas. Esse conceito de uso de *plug-ins*, que pode ser traduzido como uso de módulos de extensão. O conceito de uso de *plug-ins* permite que se possam interligar novos programas e ferramentas ao programa da plataforma já existentes, potencializando o programa existente e adicionando mais funcionalidades ao Eclipse.

O desenvolvimento da plataforma do Eclipse implica que esta tenha uma arquitetura que potencie alguma dinâmica, que permita a sua expansão e que seja adaptável a esses novos *plug-ins*. O projeto Equinox surgiu com o objetivo de dinamizar e simplificar essa integração de novos *plug-ins* com a plataforma. Para que essa integração fosse possível é necessário que ao serem adicionados novos *plug-ins* a plataforma tenha informação e acesso ao conjunto de funcionalidades, variáveis e ferramentas desse novo *plug-in*. A plataforma evolui no sentido de dinamizar e simplificar o uso de novos, pelo que foram desenvolvidas novas ferramentas ou adaptadas ferramentas já existentes. Uma das ferramentas adaptadas pelo Equinox neste

sentido, foi a solução OSGi, esta solução permitiu que fossem criados mapas de dependências entre os *plug-ins* utilizados e informação sobre as funcionalidades, versões e outra informação. E o desenvolvimento das Dependency Injection permite que para que apesar do contínuo desenvolvimento das ferramentas estas possam ser utilizadas de forma segura e que se possa proceder ao desenvolvimento de novos programas atualizar e funcionalidades anteriormente existentes ou novas funcionalidades, sem que os programas percam a sua compatibilidade. Estes desenvolvimentos permitiram criar uma plataforma modular, dinâmica e com *plug-ins* flexíveis, adaptáveis e em contante desenvolvimento.

Com o objetivo de poderem ser criados IDE e ferramentas para o desenvolvimento de linguagens de programação, a plataforma tem de ter uma interface modular que possa ser adaptada, reorganizada e personalizada dependendo das necessidades da linguagem a programar, não apenas a interface visual, mas também a interface de utilização homem máquina. Nesse sentido foram integradas e desenvolvidas as ferramentas SWF e JFace, estas ferramentas permite a criação de botões, barras, menus e a implementação com o Workbench, permite uma interface personalizada, uma diferente "cara" e integração para o desenvolvimento de linguagens que exigem diferentes interações e disposições de janelas, menus e utilidades.

As Applications Services que se pode ver na imagem, são Interfaces de Programação de Aplicativos (APIs) que surgiram de retirar de API existentes, apenas as funcionalidades mais comuns e disponibiliza-las de uma forma simples para que os programadores que não tenham elevado conhecimentos dessas funcionalidades as possam utilizar.

2.2.2 - Eclipse JDT

O Eclipse JDT foi o projeto de ferramentas de desenvolvimento de linguagem Java criado para o Eclipse SDK. Este projeto, além de proporcionar um potente e completo IDE para o desenvolvimento de programação em linguagem Java, também permite perceber qual a potencialidade da Plataforma Eclipse, uma vez que o Eclipse JDT é também a ferramenta e o objetivo do Projeto Eclipse.

O Eclipse JDT está dividido em duas partes, o JDT UI e o JDT Core. O JDT UI é composto por um conjunto de *plug-ins* que estão focados na interface com o utilizador, estão focados na adaptação e interação do Workbench, de referir o navegador de projetos, o editor e um conjunto de ferramentas que proporcionam uma interface dinâmica, como a formatação em tempo real do código, ferramentas de procura, de comparação. O JDT Core é o conjunto de ferramentas que estão responsáveis pela análise, compilação, diagnóstico da linguagem desenvolvida e da respetiva criação dos programas e ferramentas desenvolvidos.

2.2.3 - Eclipse PDE

Eclipse PDE (*Plug-in Development Environment*) é o *plug-in* do Eclipse SDK que proporciona um ambiente de desenvolvimento de *plug-ins*. Basicamente é um conjunto de

ferramentas especializadas para a criação e desenvolvimento de *plug-ins* para a plataforma Eclipse.

O objetivo do Eclipse PDE é disponibilizar ao utilizador ferramentas de desenvolvimento de *plug-ins* para o Eclipse, permitindo que um programador sem grandes conhecimento consiga desenvolver um *plug-ins*, mas também que permita que um programador experiente possa utilizar ferramentas no sentido de criar *plug-ins* complexos e poderosos. Para isso o Eclipse PDE disponibiliza ferramentas para criar, desenvolver, testar, diagnosticar e compilar e implementar *plug-ins*.

O Eclipse PDE está dividido em 3 componentes, o PDE UI, o PDE Build e o PDE API Tools.

A componente PDE UI é a componente com mais importância e a base do desenvolvimento de todo o *plug-in*. É nesta componente que estão disponíveis as ferramentas para criação, edição e desenvolvimento de um novo *plug-in*, tanto o desenvolvimento do core do programa como da interface com o utilizador. Possui ferramentas para edição dos programas e criação das dependências das diferentes funcionalidades e ferramentas assistentes para um desenvolvimento assistido e mais simples da interface do novo *plug-in*, quer ao nível visual, quer ao nível da usabilidade e interação, e ferramentas assistentes para uma melhor utilização de funcionalidades já existentes, na assistência da adaptação dessas funcionalidades.

Depois de estar desenvolvida uma parte ou a totalidade funcional do programa tem-se a componente PDE Build, que é a componente responsável pela compilação do código e implementação e criação dos *plug-ins*, funcionalidades ou aplicações. A componente de criação de *plug-ins* permite seleção de diversas opções na criação dos *plug-ins*, porem este assunto não será discutido neste documento.

A componente PDE API Tools é a componente que permite aos programadores obter relatórios de manutenção dos seus programas desenvolvidos. Permite que os programadores obtenham relatórios de erros dos seus programas a diferentes níveis, desde erros de versões, incompatibilidade de *plug-ins*, erros do próprio programa que não tenham sido detetados anteriormente.

O Eclipse PDE será uma das principais ferramentas que serão utilizadas na realização da dissertação, uma vez que este é o principal objetivo da dissertação, desenvolver uma IDE para as linguagens de programação da norma IEC 61131-3.

2.2.4 - Graphiti e GMF

A norma IEC 61131-3 define 5 linguagens de programação sendo que duas linguagens são textuais e as restantes 3 linguagens são gráficas. Está previsto que no projeto deverá ser centrado no desenvolvimento das linguagens textuais e uma linguagem gráfica, a qual está prevista que será a ST.

A existência de uma linguagem gráfica a desenvolver no IDE implica que o IDE a desenvolver terá de ter funcionalidade de criação e edição gráfica.

O Eclipse SDK possui uma ferramenta que surgiu com o objetivo de proporcionar a criação de editores e de janelas de visualização gráficos de elevada complexidade, o projeto Graphical Editing Framework (GEF). Devido à sua complexidade e potencialidade a aprendizagem desta ferramenta é complexa e o tempo de aprendizagem é elevado pela que o projeto de uma linguagem gráfica seria muito moroso para o tempo de dissertação.

No entanto, surgiram outras ferramentas que integrando duas ferramentas base permitem a criação de editores e janelas de visualização gráficas de forma mais simples e com aprendizagem mais rápida. Duas dessas ferramentas são, a Graphical Modeling Framework (GMF) e a Graphiti (Graphical Tooling Infrastructed).

Estas ferramentas têm por base duas ferramentas a GEF, acima referida que proporciona uma complexa e potente forma de criar e desenvolver editores e janelas de visualização gráficas, e a ferramenta Eclipse Modeling Framework (EMF), esta ferramenta permite a criação de modelos e geração de código partindo de um modelo de dados estruturado para a criação de ferramentas e novas funcionalidades.

A GMF é uma ferramenta que tem como princípio um modelo de definição gráfico, separando o grafismo do modelo a utilizar da sua definição é possível utilizar a ferramenta GEF para potencializar a visualização e interação do editor, mantendo uma programação mais simples e modular com a ferramenta EMF. É possível também reaproveitar os modelos gráficos para outras definições, ou reutilizar essas definições em outros modelos gráficos. A ferramenta GMF permite assim, o desenvolvimento independente de definições e modelos, sendo que posteriormente através de um mapeamento entre os modelos e definições se podem gerar as funcionalidades dos diagramas. Este conceito permite ter as funcionalidades distribuídas e uma grande flexibilidade no desenho elementos gráficos, no entanto, implica que sempre após cada alteração tem de ser ajustado o mapa e o diagrama tem de ser recriado e apesar de não ser necessário um conhecimento tão profundo do GEF, tem de se ter algum conhecimento básico.

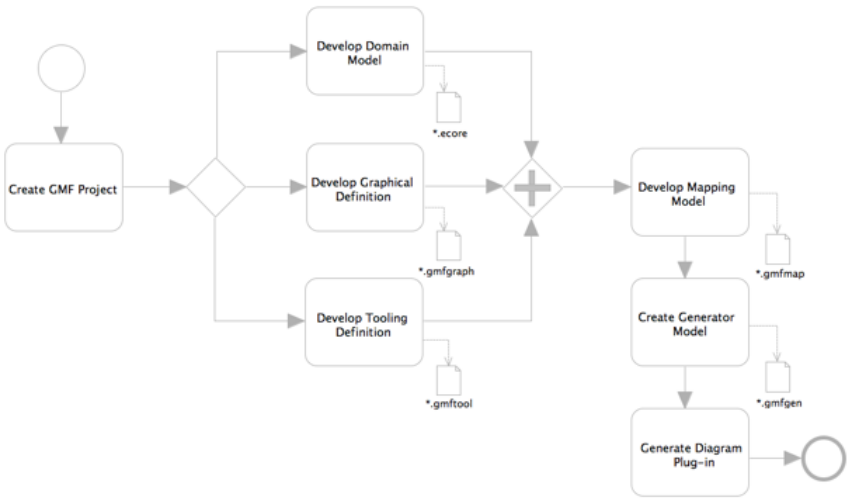

Figura 2.8 - Arquitetura do GMF.

A ferramenta Graphiti usa as mesmas ferramentas de base, a GEF e a EMF, no entanto apresenta um conceito diferente do conceito da GMF. Utiliza as funcionalidades da ferramenta GEF para o desenvolvimento dos elementos gráficos e a ferramenta EMF na estrutura e definição dos modelos, e tem uma arquitetura vocacionada para a interação durante a execução. Dividindo uma componente de interação e um motor de renderização na interface e tendo um processo de execução separado da interface, permite que este processo, identifique o elemento em questão, verifique a funcionalidade prevista para essa interação nos modelos definidos (Domain Model) e replica essa funcionalidade no modelo em imagem que será renderizado pelo motor.

O Graphiti permite assim que a definição das funções seja centralizada no Domain Model e seja mais flexível pois não é necessário realizar o mapeamento e criação dos elementos gráficos. Possui também a vantagem de não ser necessário conhecimento da ferramenta GEF.

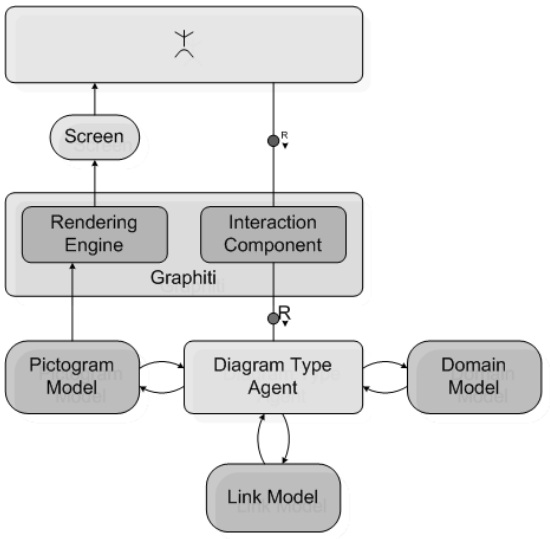

Figura 2.9 - Arquitetura do Graphiti.

2.3 - IDE existentes para a norma IEC61131-3

Atualmente já existem IDE para a criação de projetos da norma IEC 61131-3. Nesta secção serão apresentados alguns dos IDE, que são considerados mais próximos do IDE resultante do projeto alvo desta dissertação. Serão apresentados IDE livres e comerciais para uma comparação mais diversificada.

2.3.1 - CoDeSys

Um dos mais conhecidos IDE independentes que estão presentes num significante número de equipamentos industriais é o CoDeSys. O CoDeSys (acrónimo de COntroller DEvelopment SYStem) pertence à empresa 3S-Smart Software Solutions e possui as linguagens de programação padronizadas na norma IEC 61131-3.

Além do IDE da CoDeSys, esta empresa possui uma outra ferramenta interessante: o CoDeSys Control runtime environment. Este *software* pode ser instalado num PLC e, depois de instalado, permite de forma fácil a programação do mesmo equipamento através do CoDeSys IDE.

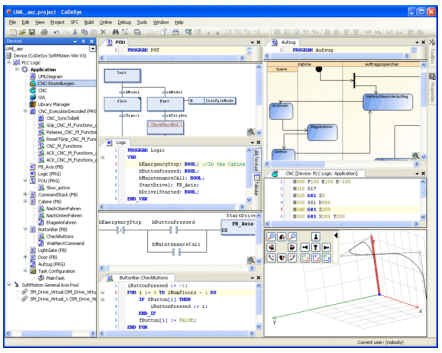

Figura 2.10 - Imagem do CoDeSys.

2.3.2 - Infoteam OpenPCS

O Infoteam OpenPCS é um IDE comercial pertencente à empresa Infoteam Software AG. É à semelhança do CoDeSys, outro ambiente de desenvolvimento integrado que suporta todas as linguagens da norma IEC 61131-3.

Trata-se de um IDE completo que possui vastas funções de visualização online e de diagnóstico dos programas assim como ferramentas de simulação.

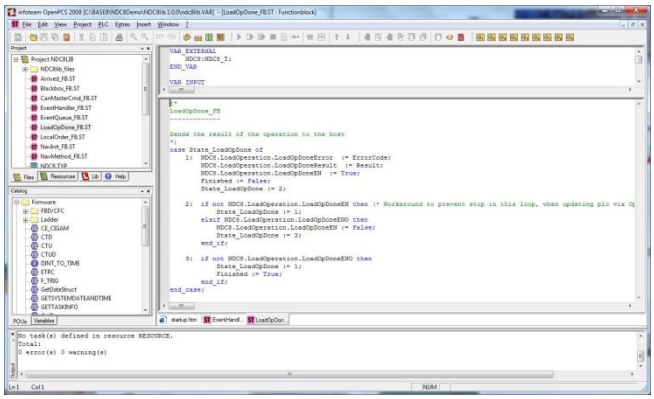

Figura 2.11 - Imagem do Infoteam OpenPCS.

2.3.3 - Beremiz Project

O projeto Beremiz, que tem o suporte da empresa francesa LOLITech, surgiu com o objetivo de criar um IDE multiplataforma das linguagens de programação respeitantes à norma IEC 61131-3. O editor gráfico foi escrito em Python o que confere ao programa a característica de multiplataforma, uma vez que permite que o código corra em diferentes plataformas como é o caso do Windows e Linux. Este projeto tem, também, a vantagem de ser um IDE livre e de código aberto que permite a participação, não apenas da equipa que criou e desenvolveu o projeto, mas que poderá ter o apoio de toda a comunidade de programadores desta linguagem e dos próprios fabricantes dos equipamentos industriais.

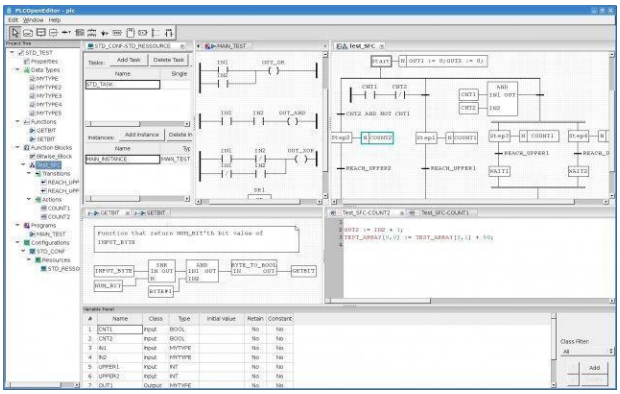

Figura 2.12 - Imagem do *Beremiz Project*.

2.3.4 - Whimori CDK: a Control Program Development Kit

Whimori CDK um projeto originário da Universidade da Coreia da Tecnologia e Educação (Korea University of Technology and Education). É um IDE baseado na norma IEC 61131-3 e nos TC6-XML schemes desenvolvido na plataforma Eclipse.

Porém, o Whimori CDK não disponibiliza todas linguagens da norma IEC 61131-3, apenas das linguagens IL e LD. No entanto, como foi desenvolvido na plataforma Eclipse, dos projetos existentes o projeto Whimori CDK é o que mais se aproxima do objetivo desta dissertação.

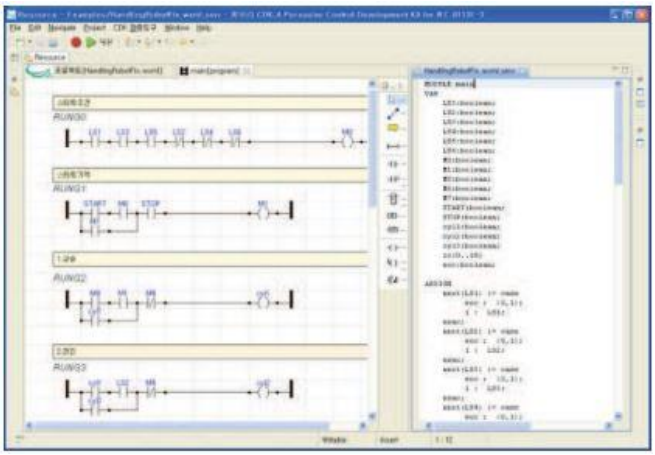

Figura 2.13 – Imagem do Whimori CDK.

Capítulo 3

Desenvolvimento

Este capítulo apresenta o desenvolvimento efetuado na realização desta dissertação. Inicialmente será apresentado o desenvolvimento realizado para a criação das variáveis necessárias na definição dos POUs (3.1 - Editor de Variáveis). Após a criação desse *plug-in* e definição de gravação e acesso dessas variáveis em ficheiros, foi necessário restruturar o *plug-in* criado na dissertação anterior para visualizar a estrutura e conteúdo do projeto para apresentar esses ficheiros (3.2 - Navegador). Nesse *plug-in* foram realizadas correções, alterações e implementações de novas funcionalidades úteis para a navegação nos projetos e para servir de interface aos restantes *plug-ins*. Foi também desenvolvido um *plug-in* para a definição de tipos de dados dos projetos (3.3 - Editor de UDT). Devido às dependências cíclicas existentes entre os diferentes *plug-ins* foi necessária a restruturação dos *plug-ins* criados no projeto que é explicada na secção 4 deste capítulo (3.4 - Reestruturação do código do projeto).

3.1 - Editor de Variáveis

Esta secção apresenta o estudo preliminar e desenvolvimento de um novo módulo de Eclipse denominado Editor de Variáveis (Figura 3.1). Este módulo funcionará como *plug-in* e terá como objetivo a criação de variáveis para o POU.

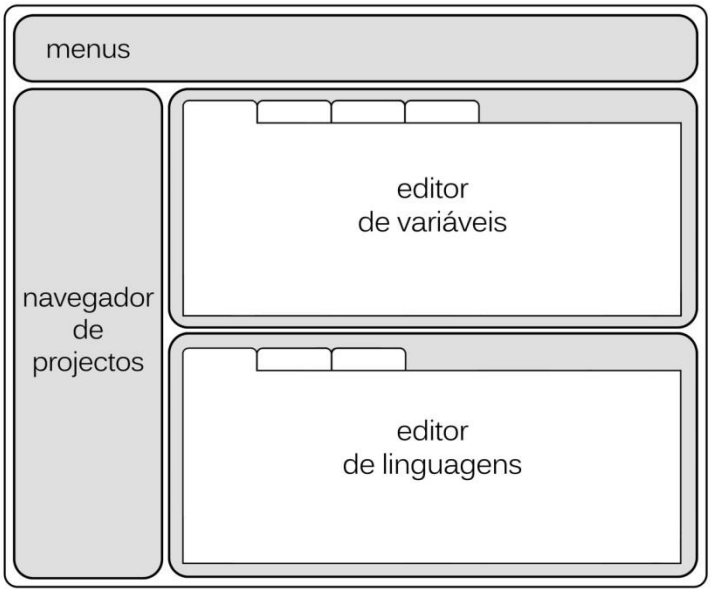

Figura 3.1 - Esboço do estudo da distribuição dos módulos existentes (**Navegador** e editor de linguagens) e do novo módulo **Editor de Variáveis**

Para estudo de funcionalidade e apresentação deste módulo foram analisadas três interfaces possíveis de apresentação do Editor de Variáveis, resultantes da pesquisa efetuada e do estudo do estado de arte existente.

A primeira interface baseia-se numa abordagem puramente textual, seguindo a especificação da norma, que resulta na criação de um editor semelhante aos editores para as linguagens textuais desenvolvidos por Filipe Ramos na sua dissertação [4](Figura 3.2). Esta abordagem apresenta uma maior liberdade, flexibilidade e é uma interface mais dinâmica. Como desvantagens apresenta o facto de ser uma interface gráfica mais limitada e de ser mais propícia à criação de erros uma vez que a estrutura e organização das variáveis fica a cargo do utilizador. A visualização desta interface seria num módulo de Eclipse separado dos módulos dos editores das linguagens da norma. Apesar de poder ser integrado com os editores textuais, esta opção foi deposta devido ao facto de criar na perspetiva visual do Eclipse comportamentos e estruturas diferentes, que causariam confusão ao utilizador.

Figura 3.2 - Primeiro interface estudado

A segunda interface analisada foi uma abordagem ao estilo de tabela, onde a navegação e a edição de variáveis se assemelharia a um programa de folhas cálculos ou a uma tabela num processador de texto (**Figura 3.3**). Cada uma das variáveis seria definida em linha e cada coluna seria um elemento da definição da variável, devidamente identificado no topo da coluna. Esta interface permite um maior desenvolvimento gráfico da apresentação que a anterior. O facto de a organização e estrutura das variáveis não depender do utilizador, resulta numa leitura mais percetível. Não tem tanta flexibilidade quanto a anterior, nem uma edição tão direta. Porém, como se trata de uma interface mais organizada e estruturada, é menos propícia a erros, permitindo um maior controlo sobre a edição das variáveis. O módulo para edição da variável seria, desta forma, visualmente diferente dos editores das linguagens textuais e gráficas, proporcionando assim uma identificação mais distinta entre os módulos.

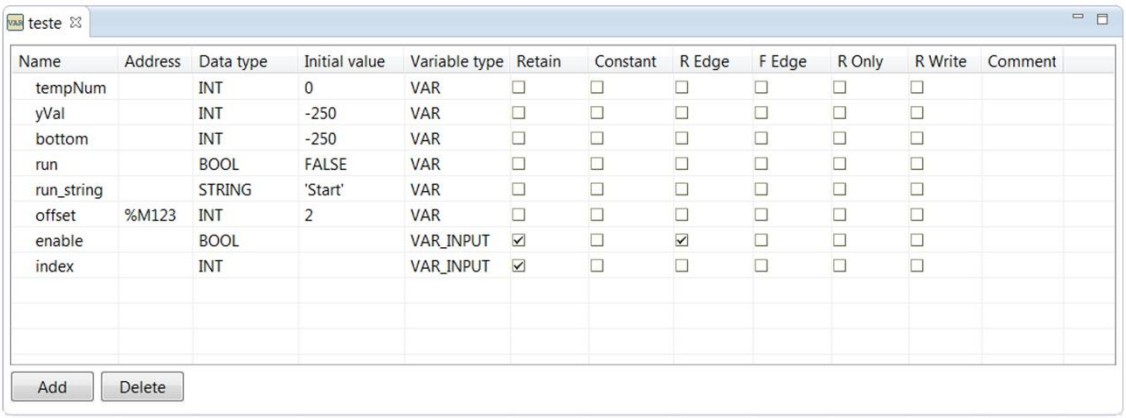

Figura 3.3 - Segundo interface estudado

A terceira interface é uma abordagem de interação diferente. Ao invés de existir uma interação direta do utilizador nas variáveis, a adição e edição das variáveis passaria a ser realizada através de janelas de diálogo. A tabela apresentaria os elementos das variáveis de forma estática com uma caixa de seleção por cada variável e os botões de interação (Figura 3.4). Quando o utilizador pretendesse adicionar uma variável apareceria uma janela de diálogo com campos em branco para serem preenchidos pelo utilizador (Figura 3.5). Caso o utilizador pretendesse editar uma variável, nesses campos apareceriam os valores existentes na variável a editar. Esta interface tem a vantagem de ter um controlo total sobre a inserção e edição de cada variável e poder proceder à validação antes da inserção, pelo que seria a interface com potencial para não existirem erros na criação de variáveis. Sendo a apresentação estática e sem interação direta seria a interface com maior potencial para ser graficamente mais apelativa. No entanto, é a interface menos dinâmica, menos flexível e com menos liberdade.

Após análise das vantagens e desvantagens de cada interface apresentado foi escolhida a segunda interface para representar o módulo o **Editor de Variáveis**.

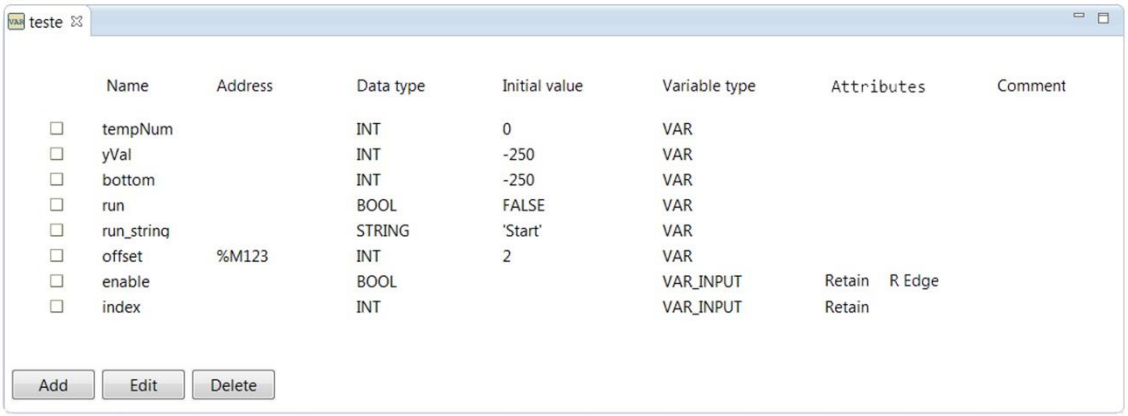

Figura 3.4 - Esboço da terceira interface

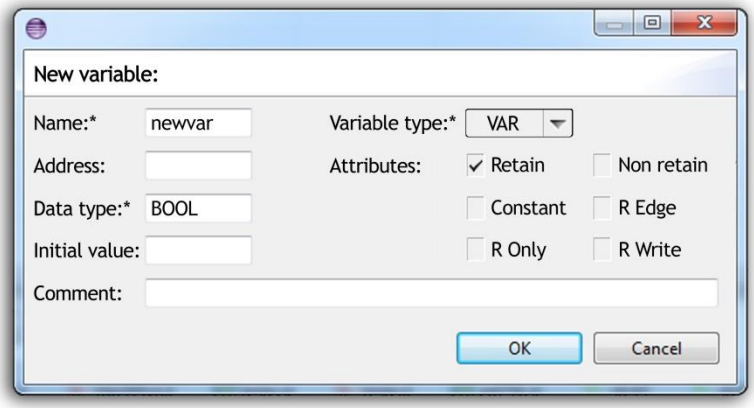

Figura 3.5 – Janela de diálogo para adição de variável (attributo Retain selecionado como exemplo).

3.1.1 - Implementação em forma de tabela

Após estudo e análise sobre a criação de uma tabela num módulo Eclipse, foi decidido usar o Interface de Programação de Aplicações ***SWT/JFace*** (API) para a criação deste módulo. Esta é uma API que nos fornece as ferramentas necessárias para a criação de interface pretendida para o **Editor de Variáveis**. Como referido, na perspetiva do Eclipse existem dois tipos de módulos: podem existir módulos do tipo visual e podem existir módulos do tipo editor. Apesar de este *plug-in* ter sido criado para funcionar como um editor optou-se por um módulo do tipo visual. Esta escolha deve-se ao facto de o módulo visual permitir uma edição gráfica mais versátil, de esse tipo de módulo permitir uma separação mais simples dos módulos dos editores (que é um objetivo referido na introdução deste capítulo) e também devido a não ser pretendido que o Editor de Variáveis tenha uma versão temporária de edição antes de ser gravada (denominada em inglês por "*dirty state*"), isto é, pretende-se que as alterações nas variáveis sejam imediatamente gravadas. Pelo que o tipo de módulo visual é o que se enquadra neste desenvolvimento.

Para a criação do módulo do **Editor de Variáveis** foi criado o *plug-in* ***VarTableEditor***. Neste *plug-in* foram desenvolvidas classes para criarem a tabela, as variáveis, e permitirem os comportamentos e funcionalidades que são pretendidos.

Para permitir que outros *plug-ins* possam chamar o Editor de Variáveis foi criada a extensão com a identificação (id) ***"org.feup.mieec.iec61131.views.VarTableEditor"***, que quando é ativada chama a classe ***VarTableEditor***, existente no *plug-in*. Esta classe é responsável pela criação de um módulo visual do Eclipse, pelo que estende a classe abstrata ***ViewPart*** que é uma base para a implementação dos módulos visuais do Workbench.

Uma das diferenças entre o módulo tipo visual e o módulo tipo editor é que neste último o módulo fica associado a um ficheiro. No módulo optado essa associação não é feita, pelo que a classe ***VarTableEditor*** é responsável por obter a informação relativa ao ficheiro que é passada na extensão. Com esta informação, a classe usa o nome do ficheiro para editar o texto que aparece no separador do módulo, para uma identificação visual dessa associação. Por fim a classe **VarTableEditor** chama a classe **VarTable**, passando-lhe também a informação do ficheiro.

A classe ***VarTable*** é a classe responsável pela criação da parte gráfica do interior do módulo, pela criação da tabela, pela criação dos botões de interação para adicionar e apagar variáveis e pela interligação com todas as funcionalidades e ações do módulo.

Decidiu-se criar uma tabela com o número de colunas correspondentes ao número de elementos e atributos possíveis de uma variável. Foram assim criadas treze colunas: as cinco primeiras colunas correspondem respetivamente ao nome, endereço, tipo de dado, valor inicial e tipo de variável; da sexta à décima segunda são colunas em que a informação corresponde aos atributos que a variável poderá ter (ser retentiva, não retentiva, contante, ser detetada à subida de flanco, ser detetada na descida de flanco, ser apenas de leitura ou ser de leitura e de escrita); por fim, a última coluna permite a inserção de um comentário ou observação à variável. De todas as colunas apenas três são de inserção obrigatória, sendo estas as respeitantes ao nome, tipo de dado e tipo de variável; todas as restantes são opcionais e podem ser desabilitadas consoante a situação, seguindo assim a definição da norma. Todas as colunas têm uma largura pré-definida, porém o utilizador poderá ajustá-la de forma livre. É também nesta classe que é definido que cada linha é uma variável, como já referido na apresentação da interface.

Nesta classe são também definidos os botões existentes no módulo de interação com a tabela. Foram criados dois botões: um primeiro botão com o objetivo de inserir na tabela uma nova linha com uma variável pré-editada contendo os dados obrigatórios; o segundo botão para apagar a linha selecionada.

3.1.2 - Navegação e edição dos elementos da tabela

O tipo de navegação dentro da tabela a interpretação de edição das células e o tipo de edição das células, também estão definidos na classe ***VarTable***. A definição do tipo de edição de célula de cada coluna foi feita considerando o grau de liberdade de escrita necessária ao conteúdo e ao número de possibilidades. Assim a primeira, segunda quarta e última colunas foram definidas como colunas com células de edição de texto simples onde são inseridos textualmente os dados correspondentes aos elementos da variável já identificados no parágrafo anterior. O tipo de edição definido para as células das colunas respeitantes aos atributos das variáveis foi uma caixa de seleção, uma vez que o seu valor é do tipo booleano (está ativo ou desativo). Relativamente à coluna do tipo de variável, foi definida uma célula com caixa de combinação, uma vez que existem um número reduzido de opções; quando o utilizador pretende editar a célula será apresentada uma caixa de combinação com as possíveis escolhas. A terceira coluna corresponde ao tipo de dado da variável; este elemento apresenta um número definido de opções elementares, porém permite também um número ilimitado de opções derivadas, pelo que foi definida uma união de uma caixa de combinação e inserção textual. A caixa de combinação apresenta uma lista das opções elementares, sendo que o utilizador poderá usar uma das opções ou editar textualmente a célula para definir o tipo de dado da variável (**Figura 3.6**).

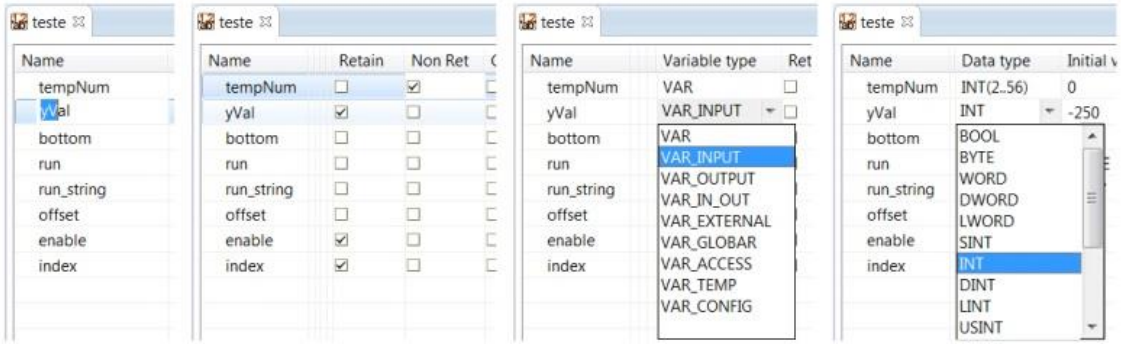

Figura 3.6 - Tipos de edição de célula, da esquerda para a direita, tipo de edição textual, tipo de caixa de seleção, tipo de caixa de combinação, união entre caixa de combinação e edição textual

A interação e navegação foram desenvolvidas para que fossem intuitivas, rápidas e simples, e o mais parecidas com os programas semelhantes. A interação com o rato foi definida e desenvolvida para que com apenas um simples clique numa célula, esta ficasse selecionada e com um duplo clique o seu conteúdo ficasse acessível à edição. Nesta dissertação designa-se por modo de edição, quando uma célula fica com o seu conteúdo acessível, quer seja uma célula com edição do tipo texto simples, edição de caixa de combinação ou edição combinada dos dois tipos referidos. As células que têm edição do tipo caixa de seleção, não apresentam modo de edição, uma vez que a alteração do conteúdo da perspetiva do utilizador é instantânea, no entanto, é considerado nesta dissertação que neste tipo de células, entrar em modo de edição é alterar o seu valor booleano. Quando se está em modo de edição de uma célula, para que o valor em edição seja inserido no elemento é suficiente que se saia do modo de edição.

Relativamente à interação com o rato no módulo ***Editor de Variáveis*** para se sair do modo de edição de uma célula podem-se fazer umas das seguintes interações, clicar numa outra célula, clicar numa zona vazia da tabela (sem linhas de variáveis), fazer duplo clique noutra célula ou clicar fora do módulo **Editor de Variáveis**. Com a primeira operação passa a estar uma nova célula selecionada, como resultado da segunda a célula que estava a ser editada passa a estar apenas selecionada, com o duplo clique noutra célula está entrará em modo de edição e no caso da última operação a célula passa a estar simplesmente selecionada, mas o módulo irá deixar de estar salientado.

O rato permite um acesso direto às células e à sua edição, enquanto, a navegação pelas células é feita recorrendo ao teclado. As setas direcionais do teclado permitem a navegação pelas células correspondente à direção do teclado, caso uma célula esteja selecionada, mas não esteja em modo de edição. Caso esteja em modo de edição, as teclas direcionais permitem a movimentação dentro da célula. Quando uma célula está selecionada ao ser premida a tecla *"enter"*, a tecla *"F2"*, a tecla *"espaço"* ou a tecla de apagar à esquerda *"backspace"* a célula entra em modo de edição, podendo ser alterado o seu conteúdo. No caso de ser uma célula com caixa de seleção é alterado o seu valor booleano, como é o caso das células correspondentes aos atributos das variáveis, como já foi referido. Também se pode usar a tecla de tabulação *"tab"* para aceder ao conteúdo da célula seguinte, mesmo estando no conteúdo de uma célula; com esta navegação caso a célula seguinte seja uma célula com caixa de seleção, o conteúdo desta não será alterado, apenas seguirá com a navegação.

Outra diferença entre as setas direcionais e a tecla de tabulação, é que caso se a célula selecionada for a célula do final de uma linha, ao premir a tecla para o lado direito esta não surtirá navegação para uma nova célula, mas ao pressionar a tecla de tabulação a navegação será feita para a primeira célula da linha seguinte. No entanto, se for na última célula da última linha não surtirá navegação para uma nova célula. Se ao pressionar a tecla de tabulação a tecla *"shift"* estiver premida a navegação será no sentido inverso, ou seja, ao invés de seguir para a célula seguinte, esta irá para a célula anterior. Ao inverso da navegação por tabulação sem a tecla *"shift"*, caso a célula selecionada seja a primeira célula de uma linha, a navegação salta para a última célula da linha anterior, à exceção da primeira linha, que não tem linha anterior pelo que não surtirá navegação para uma nova célula.

Foi descrita a forma de sair do modo de edição através da interação do rato. Também é possível sair do modo de edição com o teclado, existindo para isso três formas de o efetuar, uma já foi descrita que é o uso da tecla de tabulação, que já foi referido. Uma segunda forma de sair do modo de edição é premindo a tecla "*enter*", ao premir esta tecla o texto que está a ser editado será inserido no valor da célula. A última forma de sair do modo de edição é premindo a tecla "*esc*", ao premir esta tecla o valor da célula não será alterado independentemente de ter sido feita alguma alteração ao texto na edição. Funcionando como um cancelamento da edição.

Para permitir uma interação mais rápida com a tabela existem funcionalidades para adicionar variáveis através do teclado que são possíveis quando está selecionada uma célula, mas a célula não está em modo de edição. Se for premida a tecla *"enter"* com a tecla de controlo *"ctrl"* já premida, esta junção de teclas irá inserir uma nova variável a seguir à variável selecionada. Se além da tecla de controlo estiver também premida a tecla *"shift"* será inserida uma nova variável na linha anterior à linha da célula selecionada. Se for premida a tecla de apagar à direita *"delete"* a linha selecionada será apagada, caso não seja única. As mesmas funcionalidades de adição e remoção de variável da tabela existem no menu de contexto quando se clica com o botão do lado direito do rato numa célula.

3.1.3 - Classes Var e VarList[2]

Como já referido, a informação presente em cada linha da tabela representa a informação de uma variável. Para organizar essa informação inserida na tabela foram desenvolvidas duas classes: a classe ***VarList*** e a classe ***Var***. A classe ***Var*** foi criada para representar cada variável inserida em cada linha de tabela. Nesta classe foram criados membros de instância da classe para definir cada um dos elementos da variável da tabela. Para definir os atributos da variável, foi considerado apenas um membro de instância da classe na qual cada um dos atributos corresponde a um bit desse membro. Essa classe também apresenta vários métodos para que seja possível alterar os membros, obter o seu valor e avaliar as possibilidades da variável. A classe ***VarList*** agrega todas as variáveis da tabela e funciona como interface entre a tabela e as variáveis. Esta classe tem um membro que é a lista das variáveis existentes na tabela e outros membros que mantêm a informação de identificação das variáveis e do POU a que as variáveis estão associadas.

A tabela não apresenta diretamente os membros de cada variável da classe ***Var***, mas sim um valor textual que representa o valor do membro da respetiva classe. Assim sendo, é

2 Para simplificar a explicação destas classes nesta subsecção a variável da classe será identificada por membro da classe, para que não sejam confundidas com as variáveis do projeto.

objetivo da classe ***VarList*** servir de ponte entre a edição da tabela e a edição das variáveis. No entanto, como a classe ***VarList*** não é um interface, mas sim uma classe (pois também possui informações necessárias para o funcionamento do módulo e da interligação entre a lista de variáveis e o POU) existe uma interface que se denomina ***IVarListTable*** que faz a comunicação entre a tabela e a classe ***VarList***. Esta interface apenas apresenta os métodos que são usados na interligação entre a edição da tabela e a edição das variáveis.

3.1.4 - Definição de ficheiro com a extensão *".tvar"*

Após ter sido criado o módulo **Editor de Variáveis** e implementada a sua interface, interação e a criação das variáveis, foi necessário estudar a forma como seria guardada esta informação. Inicialmente, foi colocada a hipótese de esta informação ser guardada em formato de texto ou em formato XML. O formato de texto seguiria a norma, pelo que o ficheiro teria um conteúdo semelhante ao conteúdo de texto da primeira interface estudada no início deste capítulo, ver figura aescreveronumerodafigura. O formato XML seguiria a estrutura TC6-XML definida no PLCOpen XML. No entanto, apesar da vantagem de estes dois tipos de ficheiros prepararem a informação para uma possível conversão para ficheiro textual ou preparar a informação para uma possível exportação para ficheiro XML, implicaria que no uso mais comum desta informação o conteúdo dos ficheiros teria de estar sempre a ser convertido para a classe definida. Assim, foi decidido que a informação deveria ser gravada no formato mais simples do seu conteúdo, para que o seu acesso mais usual fosse o mais rápido. Com este objetivo foi decidido que a informação a ser guardada no ficheiro deveria ser diretamente a lista dos objetos que representam as variáveis. Assim, a gravação e leitura destes ficheiros ficaram implementadas na classe ***VarList*****,** que grava diretamente no ficheiro a lista de variáveis, conjunto dos objetos da classe ***Var***. Este processo de gravar os objetos diretamente em ficheiros é denominado por serialização, do inglês ***serialization***. Assim foi necessário implementar a interface ***Serializable*** nas classes dos objetos que se pretendia gravar diretamente no ficheiro. A título de exemplo a classe ***Var*** teve de implementar este interface, pois são os objetos desta que são gravados diretamente no ficheiro.

A extensão escolhida para estes ficheiros foi ***".tvar"*** (**t**abela + **var**iáveis). O nome do ficheiro terá o nome do POU respetivo e a localização para a gravação do ficheiro é a mesma onde se encontra o ficheiro do editor de linguagem. Assim, ambos os ficheiros (o ficheiro com o conteúdo da linguagem e o ficheiro com o conteúdo da lista de variáveis) têm a mesma localização, o mesmo nome e extensões diferentes.

3.2 - Navegador

O *plug-in* ***IEC61131_3.Navigator*** foi desenvolvido por José Ferreira [5] com o objetivo de proporcionar à perspetiva do *plug-in* ***IEC61131_3*** um módulo visual de navegação dos projetos IEC61131_3, denominado nesta dissertação por **Navegador**. Este *plug-in* apresentava apenas os projetos abertos que contenham a natureza de um projeto criado pelo *plug-in* ***IEC61131_3***. Como foi criado um novo tipo de ficheiro para guardar a lista de variáveis do POU, resultante do **Editor de Variáveis**, foi necessário proceder à adaptação deste *plug-in* anteriormente criado. Foram, desta forma, incluídos nos objetivos desta dissertação a adaptação e melhoramento deste *plug-in*.

Para apresentar o ficheiro de extensão *".tvar"* (que guarda a lista de variáveis) no **Navegador** foi necessário alterar a classe ***IEC61131_3.ProjectView*** para que esta incluísse também esses ficheiros na visualização. Foi necessário assim, alterar o método ***initialize()*** para que considerasse este novo tipo de ficheiro, criando a sua representação como objetos da classe ***TreeObject***, incluindo-os nos objetos da classe ***TreeParent*** respetiva. O método ***initialize()*** foi também alterado para que, ao invés de apresentar apenas os projetos abertos de natureza IEC61131_3, fossem apresentados também os projetos fechados, bem como outros projetos que estejam abertos mesmo não sendo da natureza identificada. Apesar de estes últimos projetos estarem abertos, uma vez que não são da natureza requerida, não apresentarão elementos filhos. Para facilitar a informação sobre o estado e natureza do projeto foram adicionados membros à classe **TreeObject**.

Para diferenciar entre projetos abertos e projetos fechados, foram usados os mesmos ícones existentes no módulo *Project Explorer* do Eclipse (que representam uma pasta aberta ou uma pasta fechada). Para diferenciar entre os projetos de natureza IEC61131_3 e os restantes foi implementado o conceito ***decorator***, que é a adição de um decorador, neste caso no ícone de identificação. Este conceito é também utilizado no módulo do *Project Explorer* para identificação de projetos Java, identificação de erros em classes, ficheiros ou *packages*.

O *plug-in* já utilizava a classe ***ViewLabelProvider*** para atribuir o ícone ao elemento da árvore de navegação consoante o tipo de elemento, por exemplo, se era um projeto, ou se era um ficheiro de linguagem do tipo ST ou IL. Na atribuição dos ícones aos elementos tipo projeto fechado ou aberto, foram feitas alterações para identificar, que caso o elemento da árvore fosse um projeto, fosse verificado o seu estado e atribuído o ícone respetivo. Para a implementação do conceito ***decorator***, foi criada uma imagem que será usada como *decorator* de identificação dos projetos de natureza IEC61131_3 e foi desenvolvido o método *decorImagebyIcon()*. Este método é chamado, quando um elemento da árvore é um projeto que está aberto e é da natureza referida, para sobrepor o *decorator* de identificação sobre o ícone atribuído ao elemento.

Para abrir os novos tipos de ficheiro *".tvar"* que existem agora na árvore de projetos foi necessário alterar o método ***run()*** da ação associada ao duplo clique (***doubleClickAction***) sobre um elemento da árvore. Esta alteração permite detetar quando o duplo clique for realizado sobre um objeto de extensão *".tvar"* e ao invés de chamar um editor para esse tipo de ficheiro, chama um módulo **Editor de Variáveis**. A chamada é feita através do método ***showview()***, que para a identificação do módulo visual e do ficheiro que esse módulo vai apresentar passa três argumentos: o primeiro argumento corresponde à identificação da extensão do *plug-in* do módulo de pretendido (neste caso é a identificação do **Editor de Variáveis);** o terceiro argumento é apenas para identificar que esse módulo depois de aberto se apresente visível; e o segundo argumento é uma identificação secundária, que também identifica o ficheiro que se está a abrir no módulo.

3.2.1 - Elemento TreeParent - POUelement

Com a adaptação da classe referida a visualização dos elementos ***TreeObject*** no módulo do **Navegador** fica confusa, uma vez que se todos os POU tiverem ficheiro de variáveis e ficheiro de linguagem, apareceram sempre dois ficheiros de nome igual e extensão diferente, apesar de todos estarem devidamente identificados com um ícone de representação do tipo de ficheiro. Analisando do ponto de vista do utilizador, também não é prático quando se

pretende aceder para programar ou visualizar um POU ter de se clicar em dois ficheiros para se abrir ambos os editores.

Para compensar a repetição de ficheiros que estão relacionados com o mesmo POU, foi criado um novo elemento na árvore de navegação. Este elemento, que aqui é denominado de ***POUelement***, é um objeto da classe ***TreeParent***, que apresenta como designação o nome dos ficheiros sem a extensão e inclui como seus filhos os objetos ***TreeObject*** que representam esses ficheiros. Na criação deste elemento na árvore de navegação foi definido que este seria identificado com o ícone que representa a linguagem e que os seus elementos filhos não teriam ícones de representação, para não sobrecarregar o **Navegador** com ícones repetidos sem informação adicional (**Figura 3.7**).

Para o desenvolvimento desse elemento foram adicionados membros na classe **TreeParent**, que identificam o tipo de linguagem usada nesse POU. Foi adicionada uma funcionalidade com este elemento, que teve como objetivo evitar o problema identificado no início desta subsecção, que era o de ser necessário fazer mais do que um duplo clique para abrir o editor de linguagem e de variáveis. Essa funcionalidade foi inserida através da alteração do método ***run()*** da ação associada ao duplo clique sobre um elemento da árvore para que, caso o elemento da árvore detetado seja do tipo ***POUelement***, ao invés de expandir para apresentar os elementos filho, este duplo clique abra os ficheiros filhos nos correspondentes editores.

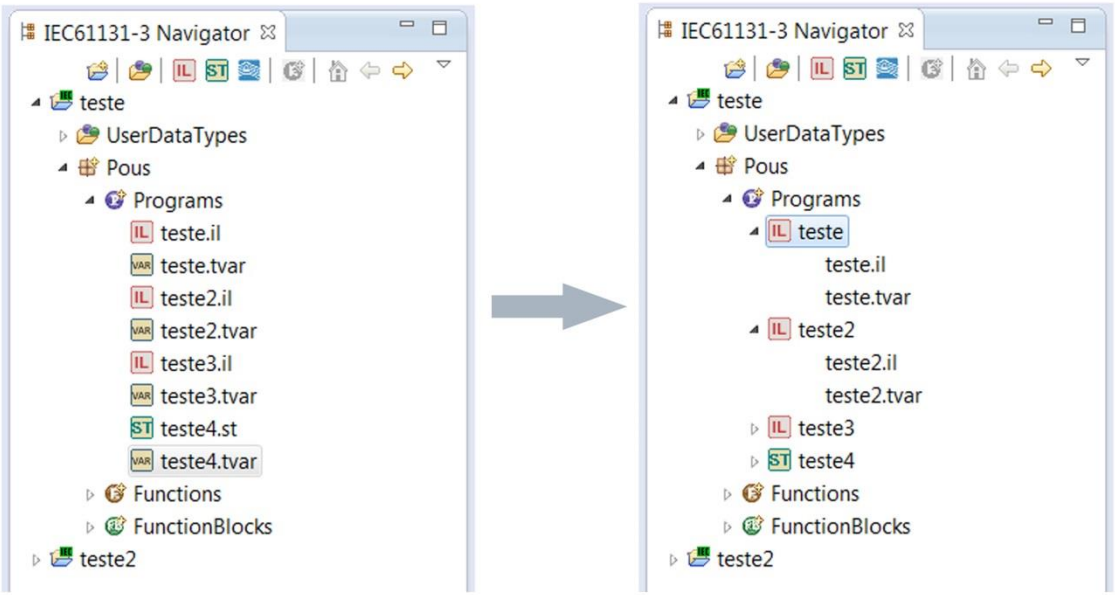

Figura 3.7- Demonstração do POUelement: à esquerda o Navegador sem POUelements, à direita o Navegador com POUelements

3.2.2 - Alteração das ações de existentes (IL, ST, SFC) e da extensão do SFC

As ações existentes no **Navegador** foram também corrigidas e melhoradas. Umas das ações que foram corrigidas foi a ação de criação de um novo ficheiro do tipo de linguagem SFC. Para a criação deste tipo de ficheiro não era possível utilizar diretamente a ação a partir do menu de contexto ou dos botões de ação existentes no **Navegador**, tendo de ser criado a partir do menu *"File"*, submenu *"New"* e escolhendo a opção *"Other"*, sendo depois necessário identificar o tipo de diagrama *graphiti* a ser utilizado[5]. Essa ação foi corrigida, sendo agora possível criar um ficheiro do tipo de linguagem SFC diretamente a partir do menu

de contexto ou dos botões de ação correspondentes à criação do ficheiro SFC do **Navegador**, sem necessidade de recorrer ao menu *"File"*.

Um problema comum às ações para criar os ficheiros de linguagens (do tipo ST, IL ou SFC) era que, independentemente de onde estava selecionada a árvore de navegação, na *wizard* de criação do ficheiro era necessário escolher de novo o projeto e a pasta onde se pretendia criar o ficheiro. Uma vez que o objetivo desta ação é simplesmente criar um ficheiro, foram alteradas as ações, para que em vez de apresentar uma *wizard*, apenas apareça uma janela de diálogo onde é solicitado o nome do ficheiro, que depois de validado é criado diretamente na localização onde está selecionada a árvore de navegação. Desta forma não é necessário voltar a escolher o projeto e a pasta onde se pretende criar o ficheiro.

Outra questão relativa ao ficheiro de diagrama SFC, que persistiu nas duas primeiras dissertações realizadas deste projecto, era a necessidade de o ficheiro guardado ter a extensão *".diagram"*. Enquanto os restantes ficheiros apresentavam uma extensão associada ao tipo de linguagem (ST e IL), os ficheiros associados à linguagem SFC apresentavam uma extensão associada ao *plug-in* base de edição dos diagramas *Graphiti*. Uma vez que o desenvolvimento das restantes linguagens de programação gráficas, poderá ser assente no mesmo *plug-in graphiti*, esta situação limitava a identificação do tipo de linguagem contida no ficheiro pela extensão. O uso do tipo de linguagem para definir a extensão de uns ficheiros e de outros não, também não seria coerente. Assim, foi alterado na criação de ficheiros SFC a extensão de *".diagram"* para *".sfc"*. Para que este tipo de ficheiro seja reconhecido foi adicionada uma extensão de associação de ficheiro no *plug-in* SFC (**Figura 3.8**).

```
<extension
  point="org.eclipse.core.contenttype.contentTypes">
    <file-association
      content-type="org.eclipse.graphiti.content.diagram"
      file-extensions="sfc"
      file-names="sfc">
    </file-association>
</extension>
```

Figura 3.8- Código XML da extensão de associação de ficheiro de extensão *".sfc"*

3.2.3 - Dinamização do estado das ações e criação de novas ações

Com as alterações efetuadas nas ações de criação dos ficheiros, os ficheiros passaram a ser criados diretamente nas localizações selecionadas no **Navegador**. Desta forma, é necessário que o estado das ações esteja ativo somente quando a localização da seleção da árvore permita que sejam criados esses ficheiros. A título de exemplo, não deverá ser permitido a criação destes ficheiros na raiz de um projeto, nem num projeto que esteja fechado ou seja de outra natureza. Com esse objetivo foi criado na classe **IEC61133ProjectView** o método ***selectionChanged()***, que tem como função avaliar a seleção da árvore de navegação e habilitar ou desabilitar as ações consoante essa seleção.

A apresentação simultânea dos projetos fechados e abertos existentes no *workspace* permite agora aceder a esses projetos. Foram assim criadas duas ações que têm como objetivo abrir um projeto fechado ou fechar um projeto aberto. Uma vez que um elemento, ou pertence a um projeto aberto ou representa um projeto fechado, não faz sentido a existência de ambas as ações ao mesmo tempo. Desta forma, foi dinamizado o menu de

contexto para que a ação que está disponível seja apresentada em concordância com o elemento selecionado.

3.2.4 - Função – *return type*

Uma das diferenças entre um POU do tipo função e os restantes POU é que a função retorna um tipo de dado. Esta característica deste tipo de POU não tinha sido considerada, pelo que ainda não tinham sido definidos os procedimentos para tratar esta informação. Após análise foi definido que esta informação seria guardada textualmente num ficheiro com o nome da função e extensão *".functype"*. A obtenção da informação sobre o tipo de dado deverá ocorrer no momento da criação dos ficheiros relacionados com o POU, quando é perguntado o nome com o qual serão gravados, sendo também criado o ficheiro aqui definido com a informação obtida.

Para realizar este desenvolvimento foi necessário alterar o método **createfile()** da classe **IEC61131ProjectView** para que na criação de um tipo de ficheiro, identifique se a criação é de um tipo de função e em caso afirmativo, seja solicitado ao operador qual o tipo de dado que a função irá retornar.

Foi definido que a apresentação dessa informação seria inserida no final do nome do elemento **POUelement** exposto na subsecção 3.2.1. A descrição do **POUelement** deverá conter o nome do POU seguido de espaço, dois pontos, espaço seguindo depois da informação do tipo de dado que a função retorna (por exemplo, "mostrador : INT").

Por fim, foi criada no menu de contexto e nos botões do **Navegador**, uma ação com a descrição *"Change Func type",* para que o utilizador possa alterar essa informação ou a definir, no caso de o POU não a apresentar. Antes de alterar essa informação deverá ser validado que o tipo de dado existe para o projeto em que está localizado.

Para realizar este desenvolvimento foi necessário efetuar alterações no método **initialize()** da classe **IEC61131ProjectView,** de forma que ao serem identificados os **POUelements**, caso um **POUelement** seja do tipo função, será acrescido à descrição a sequência espaço, dois pontos, espaço, e será acrescentado o tipo de dado que a função retorna, caso exista e seja identificado.

3.3 - Editor de UDT

Como referido a norma IEC61131-3 além de permitir a criação de variáveis associadas a tipos de dados elementares e a tipo de dados derivados dos elementares (por exemplo *arrays* ou *subranges* de valores inteiros ou booleanos), permite também a criação de novos tipos de dados. Esta dissertação tem também como objetivo o desenvolvimento de um módulo para a criação desse novo tipo de dados, também chamados de **t**ipos de **d**ados do **u**tilizador e aqui tratados por UDT (do inglês **U**ser **D**ata **T**ypes).

3.3.1 - Reutilização do *plug-in* **VarTableEditor**

Os elementos básicos de um UDT são semelhantes aos elementos básicos de uma variável. Uma primeira abordagem para a criação do módulo, poderia ser o uso do mesmo *plug-in* desenvolvido para a criação das variáveis, escondendo as colunas que não seriam necessárias

para a criação de UDT. No entanto, esta abordagem não foi seguida, pois a funcionalidade e características de variável e de UDT são diferentes. Por exemplo, pode-se ter o mesmo nome em variáveis diferentes se estas estiverem configuradas em POU diferentes, porém não se pode ter dois UDT com o mesmo nome, mesmo que em ficheiros separados. Considerando esta situação, seria necessário usar duas bases diferentes (uma para variáveis e outra para UDT) na mesma tabela ou ter apenas uma base, mas com condicionalismos na classe da base que possibilitasse a diferenciação na própria classe. Estas definições complicariam demasiado o desenvolvimento e restringiriam mais a adaptação do *plug-in*.

Embora a abordagem referida anteriormente não tenha sido seguida para a criação deste editor, denominado nesta dissertação por **Editor de UDT,** é lógico reutilizar o *plug-in* desenvolvido para a criação do **Editor de Variáveis** pelas semelhanças que estes apresentam. Por isso, foi criado um novo *plug-in* copiado do *plug-in* anterior, para ponto de partida, onde depois foram feitas as alterações necessárias para o adaptar ao novo tipo de informação.

A primeira alteração realizada no *plug-in* foi a alteração dos nomes das classes e seus membros substituindo, onde existia a sequência de letras “var” por “udt” (independente da sua capitulação). Esta alteração foi realizada com o intuito de, embora diferentes, facilmente se associar a classe ou membro de um *plug-in* ao outro.

Depois de alterados os nomes foi alterado o id do *plug-in* para "**org.feup.mieec.iec61131.views.UDTTableEditor**", uma vez que, o id do *plug-in* é a identificação do módulo pela qual é chamado, este teria de ser diferente.

Os UDT apresentam menos elementos a definir, assim foi necessário adaptar a nova classe **UDTTable**. Devido a neste editor, apenas ser necessário as colunas correspondentes aos elementos nome, tipo de dado, valor inicial e comentário, foi diminuído o número de colunas. Tendo sido as restantes colunas eliminadas da classe, assim como os membros de classe que estavam relacionados com os elementos correspondentes às colunas, e que por isso ficaram obsoletos.

A navegação, interação e o tipo de edição de células foram mantidas, pelo que os métodos respeitantes a essas funcionalidades mantiveram-se inalterados, tendo sido apenas feita adaptação das descrições para se referirem aos UDT em vez das variáveis.

3.3.2 - Classes UDT e UDTList

Como referido no início da secção anterior, as variáveis apresentam características e número de elementos diferentes dos UDT, assim foi necessário restruturar a classe **UDT** para que representasse o tipo de dado definido pelo utilizador, com os membros respetivos.

Apesar de esta classe possuir menos elementos que a classe **Var**, esta classe é mais complexa. O motivo desta complexidade é o elemento tipo de dado. Enquanto na classe **Var**, este elemento é representado diretamente por um simples identificador de um tipo de dado (elementar ou definido pelo utilizador) ou por um *array* de um tipo de dado. Na classe UDT este elemento pode representar diferentes informações de diferentes formas. A título de exemplo, poderá ser uma enumeração, que irá apresentar um número de identificadores filhos ou poderá ser também uma estrutura que apresentará um conjunto de elementos filho que serão novos UDT.

O facto de um UDT pode ser do tipo estrutura, que indica que este pode ter um conjunto de elementos filho, que são novos UDT, foi necessário adaptar a tabela para que este subelemento do UDT pudesse ser apresentado e editado. Para que essa apresentação fosse

possível, foi necessário alterar também a classe **UDTList**, que além de ter sido modificada para ser realizada a adaptação da alteração das variáveis para os UDT, foi também desenvolvida para ser inserida esta nova funcionalidade.

3.3.3 - UserDatatypeDef e subclasses

Como referido, o elemento tipo de dado do UDT poderá conter formas e informações diferentes, dependendo do tipo de derivação que tiver. Para poder reter esta informação de maneira a ser reutilizada de uma forma dinâmica, foram criadas classes para definir cada tipo de derivação e conter os seus elementos. No entanto, uma vez que todas estas classes, apesar de em formas diferentes, representarem o mesmo elemento (tipo de dado), foi necessário criar a classe abstrata ***UserDatatypeDef***. As classes, criadas para definir os diferentes tipos de derivação do tipo de dado, estenderão esta classe que funcionará como esqueleto, e onde estão definidos os elementos básicos a todas as classes seguintes, assim como métodos, que independentemente da forma do tipo de dado, terão de ser implementados.

Foram criadas as seguintes classes para definição da derivação de tipo de dado: ***Arrayof*** para armazenar o tipo de dado *array*, nesta classe será retida a informação das dimensões do e do tipo de dados base desse *array*; ***Subrange*** onde serão armazenados os limites do intervalo e o tipo de dado base desse intervalo; ***Enumerated*** onde serão armazenados os elementos da enumeração; e a classe ***Struct*** onde serão guardados os elementos UDT da estrutura. Para facilitar a definição arbitrária do número de dimensões classe ***Arrayof*** foi criada uma classe ***Dimension***, na qual é definida a estrutura de dimensão.

3.3.4 - Expressões regulares – validação textual

Para ser possível efetuar a validação do valor textual inserido na tabela, foi necessário criar ferramentas que conseguissem interpretar esse texto de forma a puderem identificar e validar a inserção. Apesar, de no *plug-in* dos editores textuais já existirem ferramentas para a deteção de palavras e números, estas ferramentas não são eficazes no objetivo aqui pretendido. Poderiam ser utilizadas para identificar a presença da palavra *array* ou da palavra *struct*, mas nesta situação, mais do que identificação de uma palavra ou um número é necessário detetar um padrão e validar esse padrão seguindo as regras da norma.

Tendo como base esse objetivo foram usadas **expressões regulares** (também conhecido por ***regex*** da abreviatura do inglês "***reg**ular **ex**pression*") como base dessas ferramentas. Explicando de forma simples uma ***regex*** é uma sequência de caracteres que define um padrão de busca. Essa sequência pode definir uma palavra exata, mas também pode definir um padrão ou uma regra para letras, números, caracteres visíveis e caracteres não visíveis.

A linguagem Java já possui ferramentas de interpretação das expressões regulares e de uso dessas expressões em valores textuais. Assim, não terão de ser utilizadas ferramentas externas nem terá de ser desenvolvida uma ferramenta para esse fim. O uso desta ferramenta apresenta duas vantagens neste objetivo. A primeira, como discutido até aqui, é poder identificar se um valor textual representa um determinado padrão, o que é útil para validar se a inserção corresponde ao que está a ser testado. A segunda vantagem é poder procurar um padrão repetidamente e usá-lo sempre que este for encontrado. A título de exemplo, um

array pode ter múltiplas dimensões, onde as dimensões são separadas por virgulas e cada dimensão é definida por dois números separados por dois pontos finais (ex: 3 dimensões: "1..5,1..7,0..3"); se for usado um padrão para detetar uma dimensão, esse padrão poderá ser usado para correr toda a cadeia textual e extrair cada dimensão desse *array* iterativamente.

Para a deteção e validação do tipo de dado é necessário que primeiramente seja possível realizar a deteção e validação dos elementos mais básicos da linguagem da norma IEC61131-3. Assim foram criados os padrões de identificação dos elementos que vão desde os identificadores, passando pelos números inteiros, números reais, representação literal de valores, até à deteção de tipos de dados como *arrays*, *subranges* e enumerações, respeitando as regras da norma (por exemplo, o facto de a capitulação das letras não ser significativa e o facto de nos números permitir a existência de *underscore* entre os algarismos). Os padrões correspondentes aos elementos mais básicos foram agrupados na classe ***Elementarchecks***, onde foram também criados métodos para testar se uma cadeia de texto corresponde a um desses padrões básicos.

Os padrões de identificação do tipo de dado foram criados nas classes respetivas, descritas na subsecção anterior, assim como, também foram criados os métodos para validar se uma cadeia de texto corresponde ao tipo de dado da classe.

3.4 - Reestruturação do código do projeto

No desenvolvimento do *plug-in* referido na secção anterior foram encontradas dependências cíclicas entre os *plug-ins*. A dependência cíclica colocava-se devido ao facto de: o ***Editor de UDT*** necessitar de aceder ao **Navegador** para obter informação dos restantes UDT existentes no projeto; o **Editor de Variáveis** necessitar de aceder ao **Navegador** para aceder aos projetos, para por sua vez aceder aos UDT, para os utilizar no elemento tipo de dado; e o **Navegador** por sua vez necessitar de aceder aos UDT para verificar os tipos de dados que poderá ter no retorno de uma função (Figura 3.9)

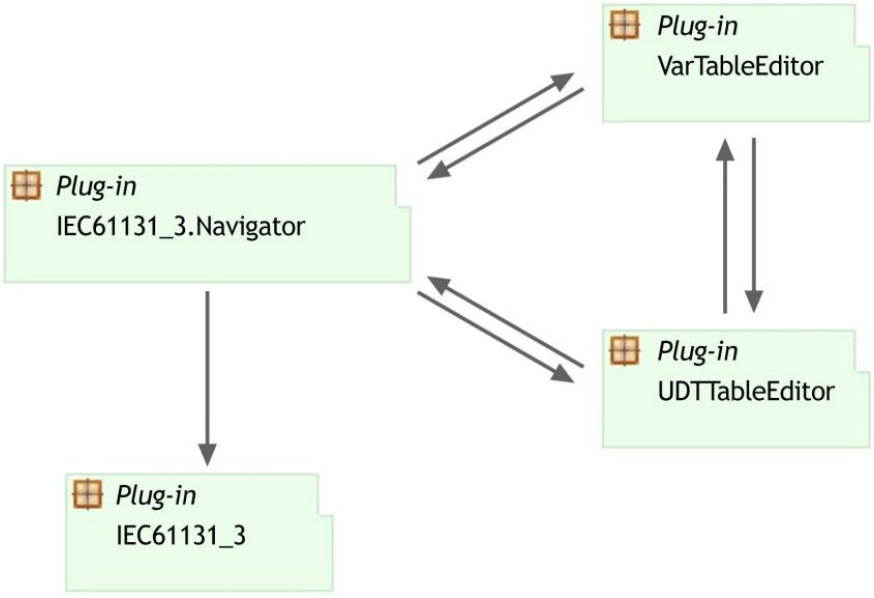

Figura 3.9 - Diagramas de dependências dos novos *plug-ins* e do *plug-in* do IEC61131_3.Navegator.

O motivo pelo qual os novos *plug-ins* acederem ao Navegador, é que este *plug-in* acede a todo o conteúdo do projeto de natureza IEC61131 criado pelo utilizador. Os novos *plug-ins* desenvolvidos acedem assim, ao **Navegador** para obter informação de forma mais rápida sobre quais os ficheiros existentes nos projetos e o seu conteúdo, uma vez que necessitam dessa informação.

Uma vez detetada a dependência cíclica entre os *plug-ins* e analisada a situação foi adicionado à dissertação o objetivo de restruturar os *plug-ins* e o código do projeto, de forma a eliminar as dependências cíclicas e melhorar o projeto.

Uma abordagem que aparenta ser mais simples e que poderia ser mais rápida, na resolução da dependência cíclica entre *plug-ins*, seria implementar parte do código do **Navegador** nos novos *plug-ins*. Se estes corressem a diretoria dos projetos para obter os ficheiros e assim identificar e obter a informação que necessitam, não teriam de aceder ao Navegador o que eliminaria a dependência cíclica existente entre estes. Depois teria de ser analisada a próxima dependência cíclica, caso existisse, e teria de ser feito o mesmo procedimento. No entanto, esta abordagem é uma simples forma de contornar o problema, mas não é uma solução, e pode mesmo não apresentar solução, pelo que esta abordagem não foi considerada.

Nesta secção será tratado assim em pormenor da restruturação do código e das classes do projeto.

3.4.1 - Análise das dependências entre *plug-ins*

No sentido de efetuar essa restruturação, foram estudadas as dependências entre todos os *plug-ins* e posteriormente entre as classes dos *plug-ins* que fossem necessários. Na **Figura 3.10**é possível ver o diagrama de dependências entre os *plug-ins* existentes antes do desenvolvimento documentado nesta dissertação.

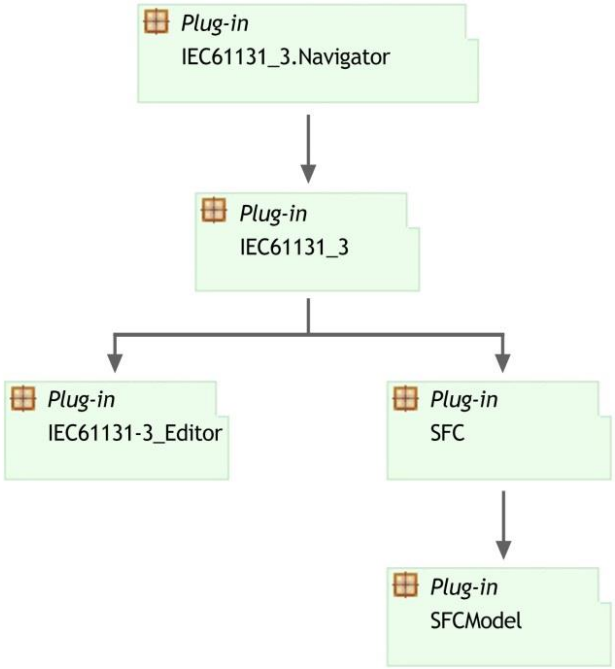

Figura 3.10 – Diagramas de dependências dos *plug-ins* no início da dissertação.

Na **Figura 3.11** está representado o diagrama de dependência de todos os *plug-ins* antes da restruturação do projeto.

Na dissertação anterior os *plug-ins* **SFC** e **Editor** foram adicionados às dependências do *plug-in* **IEC61131_3** para que estes fossem incorporados no arranque do projeto. No entanto, o **IEC61331_3,** que é o *plug-in* que permite a criação de projetos da natureza **IEC61131**, não tem dependência direta desses dois *plug-ins* referidos, o que liberta esta dependência.

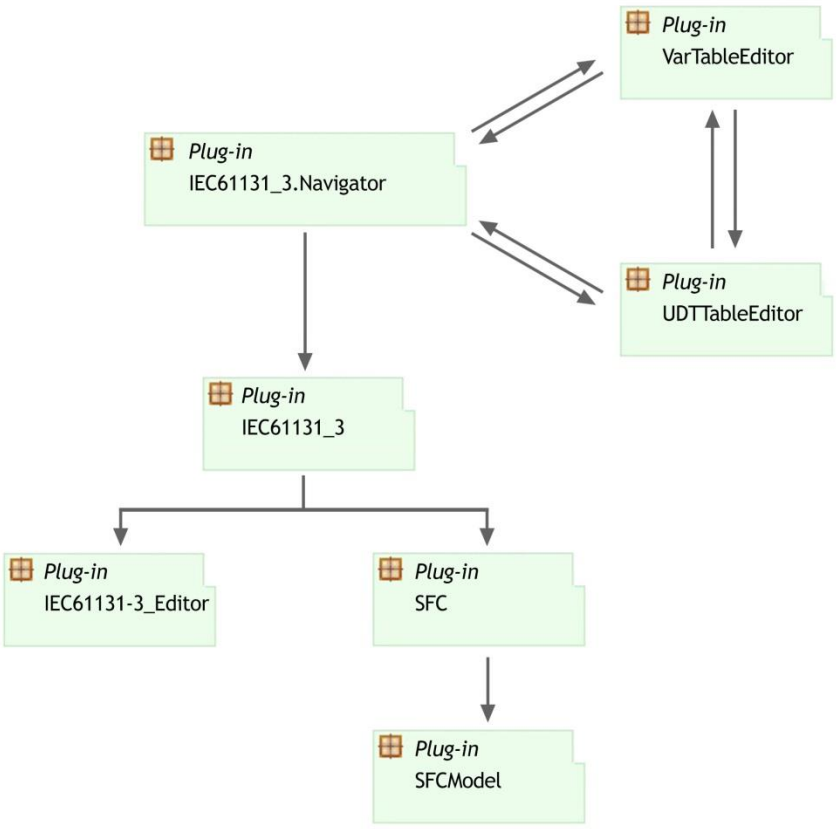

Figura 3.11 – Diagrama de dependências dos *plug-ins* antes da reestruturação.

Para restruturação do projeto foi definido que haveria um *plug-in* que funcionaria como núcleo do projeto. O objetivo de criar um *plug-in* núcleo é que todas as classes estruturais do projeto global sejam desenvolvidas neste *plug-in*, permitindo assim acesso aos *plug-ins* restantes.

O *plug-in* **IEC61131_3** é o *plug-in* que possui a informação da natureza e que é responsável pela criação dos projetos do utilizador, pelo que foi escolhido este *plug-in* para funcionar como núcleo. Todas as classes que tenham informação global ou definições elementares que poderão ser usadas por vários *plug-ins* deverão ser desenvolvidas ou deverão no final do desenvolvimento ser movidas para o **IEC61131_3**.

O *plug-in* **IEC61131_3.Navigator** é o *plug-in* que cria o módulo que funciona de interface ao utilizador para acesso, para navegação e para criação de projetos e de ficheiros da norma IEC61131-3. Devido a este *plug-in* ser a primeira linha de contacto entre o utilizador e o programa, foi definido, que este *plug-in* terá todos os restantes *plug-ins* nas suas dependências de forma que sejam incorporados no projeto.

Por fim, decidiu-se uniformizar os nomes de todos os *plug-ins*, de forma que, a sua denominação siga uma estrutura coerente. Todos os *plug-ins* usaram como prefixo o nome do *plug-in* núcleo seguido de um ponto, à exceção dele próprio (**"IEC61131_3").** Também foi decidido que o nome do *plug-in* **SFCModel** passou a ser **IEC61131_3.SFC.Model** e que o nome do *plug-in* **IEC61131-3_Editor** passou a ser **IEC61131_3.TextualEditors**.

A **Figura 3.12** apresenta o diagrama de dependências dos *plug-ins* definido para a restruturação já com os nomes dos *plug-ins* atualizados.

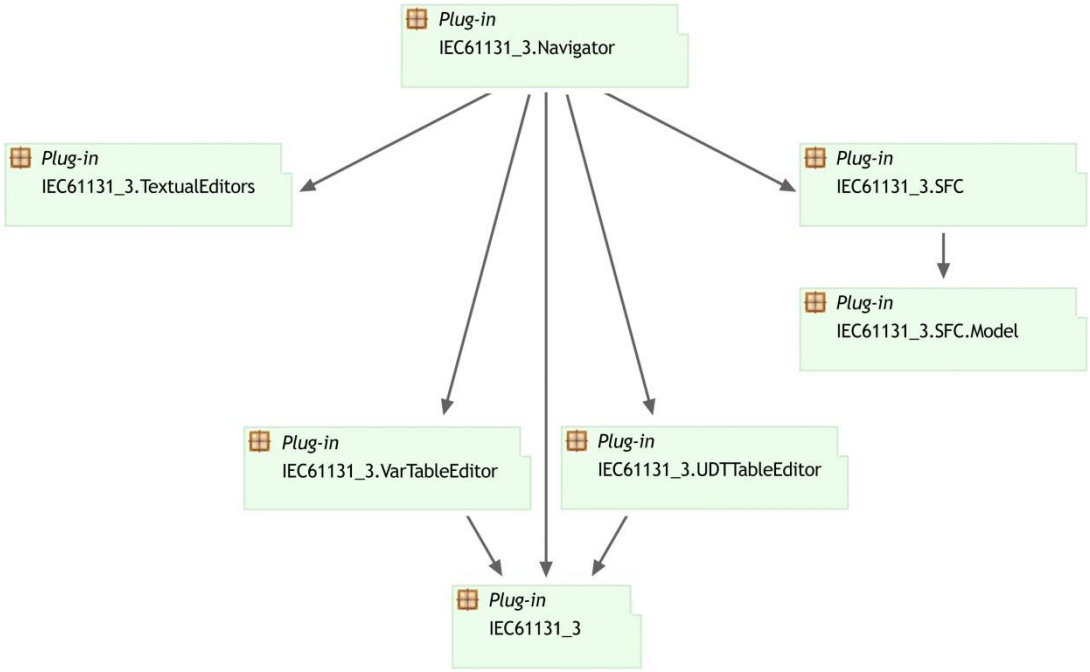

Figura 3.12 – Definição do diagrama de dependências dos *plug-ins* depois da reestruturação.

3.4.2 - Árvore do navegador: TreeProjects, TreeParent e TreeObject

A primeira informação global e que foi por isso passada para o núcleo do projeto (*plug-in* **IEC61131_3**) é a estrutura da árvore do navegador. Esta informação é um elemento da classe **TreeParent**, que como definido por José Ferreira na sua dissertação é "um nó invisível que funciona como raiz, sendo todos os projetos da norma adicionados como "filhos"."[5].

No sentido de mover essa informação para o núcleo, foi criada a classe **TreeProjects.** Esta classe tem uma variável estática da classe **TreeParent** que representará a árvore do navegador. Essa é uma variável estática (também definida por variável de classe) para que haja apenas uma instância dessa variável por todo o projeto, com o objetivo de que a informação seja uniforme.

Para a nova classe criada foi necessário extrair da classe **IEC61131ProjectView,** do *plug-in* **IEC61131_3.Navigator,** os métodos e variáveis existentes que estavam relacionados com a variável *parent*. Esta variável era o objeto que correspondia à estrutura da árvore de navegação. A título de exemplo, os métodos *initialize()* e *NavViewerAddProject()* foram dois dos métodos que extraídos e adaptados na nova classe.

Com esta alteração, foi necessário mover as classes **TreeObject** e **TreeParent** também para o núcleo, uma vez que estas duas classes são as classes que definem os elementos base da estrutura da árvore de informação.

No final, foi adaptada a classe **IEC61131ProjectView**, para que ao invés de criar uma nova árvore de navegação, crie um elemento que será um objeto da nova classe criada, onde terá a informação necessária para apresentar a árvore de navegação.

Com esta adaptação, o *plug-in* **IEC61131_3.Navigator** continua a apresentar ao utilizador a mesma árvore de navegação, com o mesmo aspeto e com as mesmas funcionalidades sem ter de fazer a gestão da estrutura da árvore de navegação, que está agora sob o *plug-in* **IEC61131_3**.

3.4.3 - Classes Var, UDT e classes elementares

As variáveis e os UDT da norma IEC61131-3, por definição, serão elementos que terão um uso global no projeto. Por exemplo, como referido no início da secção, os UDT já têm uso no *plug-in* do **Editor de Variáveis** e no **Navegador**, e a variáveis, apesar de ainda não acontecer, poderão ser usadas pelos editores das linguagens, para identificar quando o utilizador insere uma. Pelo que a passagem destas classes, bem como as classes elementares que as acompanham, enquadram-se no objetivo de centralizar no núcleo as classes e definições elementares.

Nesse seguimento, a classe **Var** do *plug-in* do **Editor de Variáveis** foi movida para o núcleo. Com a classe **Var,** foram também movidas para o núcleo as enumerações **VarTypes** e **Elementary**, uma vez que, estas enumerações são elementos da classe Var.

Do *plug-in* **Editor de UDT**, foi objetivo mover para o núcleo a classe **UDT**, bem como todas classes que a classe **UDT** tem dependências, como a classe **Elementarchecks**, **Arrayof**, **Subrange**, **UserDatatypeDef**, entre outras.

O **Editor de Variáveis** e o **Editor de UDT** continuam com os mesmos objetivos de criação, alteração e eliminação dos seus elementos. O objetivo da passagem das classes elementares para o núcleo do projeto é que, os restantes *plug-ins* possam aceder aos ficheiros de extensão *".tvar"* e *".tudt"* e consigam ler o seu conteúdo, para obter assim a informação sem necessidade de chamar o *plug-in* de edição.

3.4.4 - TreeIECProject: TreeParent + UDT

Os UDT são criados para poderem ser usados em qualquer estrutura do projeto que necessite de usar esta informação, ou seja, que possa usar os denominados tipos de dados derivados que são definidos pelo utilizador. Mesmo estando em ficheiros diferentes, os UDT não podem usar identificadores iguais, pois a sua definição é global para o projeto. Desta forma, por exemplo, ao configurar uma variável na edição do seu elemento tipo de dado, esta deve ter acesso a todos os UDT, independentemente de quantos ficheiros e UDT existam.

Com a restruturação, a classe **UDT** foi movida para o núcleo o que, como já foi referido, permite que outros *plug-ins* possam aceder aos UDT criados e gravados. Porém, não será aconselhável, por exemplo, por questões de performance, que sempre que se abre um **Editor de Variáveis** no qual se pretende alterar ou definir o retorno de uma função, o programa abra e leia todos os ficheiros do tipo UDT.

Para solucionar esta dificuldade foi criado a classe **TreeIECProject** e a classe **DatatypesList**. A classe **TreeIECProject** estende a classe **TreeParent** e tem uma variável que é um objeto da classe **DatatypesList**. E foi criada com o objetivo de substituir os elementos filho da classe **TreeProjects**, para passar assim a representar, na estrutura da árvore dos projetos **IEC61131-3**, a raiz de cada projeto. A classe **DataTypesList** foi criada para representar uma lista de tipos de dados, incluindo os elementares e os derivados onde se incluem os UDT. Foram também criados vários métodos na classe, nomeadamente para validar se a informação corresponde a um tipo de dado existente, para devolver os tipos de dados existentes, entre outras.

Na criação da estrutura definida pela classe **TreeProjects**, na criação de cada um dos elementos filho **TreeIECProject**, é efetuada a leitura dos ficheiros de extensão ".tudt"

existentes na diretoria do projeto e incluída na sua variável (objeto da classe **DataTypeList)** uma lista de todos os elementos UDT disponíveis no projeto. Com as raízes dos projetos a terem esta informação, por exemplo, o **Editor de Variáveis**, não terá de abrir e ler todos os ficheiros UDT, basta aceder a este elemento para os obter.

A classe **TreeIECProject** poderá em desenvolvimento futuro, ser usada para incluir outras estruturas que sejam globais ao projeto e tenham acesso recorrente, permitindo assim o acesso de forma mais direta e rápida.

3.4.5 - Observer

Uma das dificuldades, quando se utilizam classes para apresentar um conjunto de objetos que são criados e editados noutras classes ou *plug-ins*, é a capacidade de as primeiras manterem atualizada a informação das segundas, uma vez que pode ser modificada.

Para poder manter essa informação atualizada é possível recorrer a um ***Observer***. Um ***Observer*** (do inglês **Observer Pattern**) é um padrão de desenho de *software* que define uma típica relação de subscrição de um objeto, o subscrito, para outros objetos, os subscritores. Este *software* funciona com a implementação de uma interface que funciona como comunicador, assim quando um objeto que é subscrito sofre alguma alteração esta alteração é comunicada para todos os objetos subscritores desse objeto. Este padrão tem ainda a vantagem de poder definir quais as alterações que devem ser comunicadas e identificar o tipo de comunicação que está a ser efetuada para que o subscritor faça uma gestão correta dessa informação.

No projeto já é implementado este tipo de padrão de desenho do lado de um subscritor. A classe **TreeProjects** é um exemplo, esta classe é um subscritor da estrutura do Eclipse ***Resource*** para obter informação de quando ocorre uma alteração na estrutura de ficheiros associados ao Eclipse.

O padrão ***Observer*** será desenvolvido em classes, nas quais os seus objetos necessitem de ser subscritos. Este desenvolvimento será realizado recorrendo à utilização das classes **EventListenerList** e **ChangeListener** disponibilizadas. A classe **ChangeListener** define a criação de um objeto que observa se o objeto subscrito sofre alterações. Por sua vez a classe **EventListenerList** define uma lista de subscrições onde é inserida a subscrição do objeto criado da definição da classe anterior. Na classe onde é desenvolvido o **Observer** é ainda necessário criar pelo menos um método que identifique que alterações serão comunicadas à subscrição e dois métodos que possam ser chamados pelo subscritor, um para o subscritor poder adicionar a subscrição outro para a remover.

O desenvolvimento do lado do subscritor, à semelhança do que já é realizador no exemplo referido, é apenas necessário criar um novo objeto da classe **ChangeListener** e adicionar esse objeto ao objeto que representa a classe do subscrito, através do método para adicionar a subscrição. Na definição desse objeto da classe **ChangeListener** terá de ser definido um método que será chamado sempre que houver atualização na subscrição, com o fim de usar essa informação para atualizar o objeto, se esse for o caso.

A título de exemplo uma das classes onde foi implementado o padrão como subscrito e outra onde foi usado como subscritor, foi na classe **TreeIECProject** e na classe **VarTable**. A classe **VarTable** usa um objeto da classe **TreeIECProject** para obter os tipos de dados existentes no projeto. Para que a informação que usa esse objeto esteja sempre atualizada, foi desenvolvida a classe **TreeIECProject** como subscrito e sempre que é alterado o seu

conteúdo alerta os subscritores. Na classe **VarTable** foi criada a subscrição do objeto correspondente, e o método, criado na definição do objeto da classe **ChangeListener**, é responsável por atualizar os dados da tabela. Assim, caso seja inserido na tabela um tipo de dado que não existe, por exemplo na criação de uma instância de um FB que ainda não foi criado, este será assinalado na tabela com a apresentação desse texto a cor vermelha. Porém, logo após a adição desse tipo de dado ao projeto, no nosso exemplo com a criação desse FB, a tabela irá ser automaticamente atualizada e a cor do texto passará a preto, indicando que está validado.

Capítulo 4

Testes

Neste capítulo serão descritos testes realizados nas diversas fases de desenvolvimento e serão também apresentados resultados tipo desses testes. Embora aqui sejam apresentados apenas um exemplo de cada tipo de teste, foram realizados vários testes em ordem a obter uma maior garantia do trabalho realizado.

Os testes serão apresentados com a mesma estrutura do capítulo de desenvolvimento para uma compreensão e seguimento mais fácil. Desta forma, serão apresentados inicialmente, os testes realizados ao Editor de Variáveis (secção 4.1), na secção seguinte serão expostos os resultados do desenvolvimento do *plug-in* de navegação (secção 4.2), depois serão mostrados os resultados dos testes executados ao Editor de UDT (secção 4.3), e por fim, serão descritos os testes realizados ao desenvolvimento de restruturação do código (secção 4.4).

4.1 - Editor de Variáveis

Nesta secção estão apresentados os resultados relativos aos testes efetuados ao Editor de Variáveis. Sempre que possível e para garantir uma melhor perceção dos resultados obtidos estes são apresentados com imagens. Estas imagens podem ser simples captura de ecrã ou imagens editadas com o intuito de salientar o resultado obtido.

4.1.1 - Criação do módulo visual

Na Figura 4.1 é apresentado o módulo **Editor de Variáveis** criado onde se verifica: a identificação do módulo na aba separadora; a existência de um interface do tipo tabela, sem elementos; as treze colunas e a sua descrição; e a existência de dois botões para adicionar e remover.

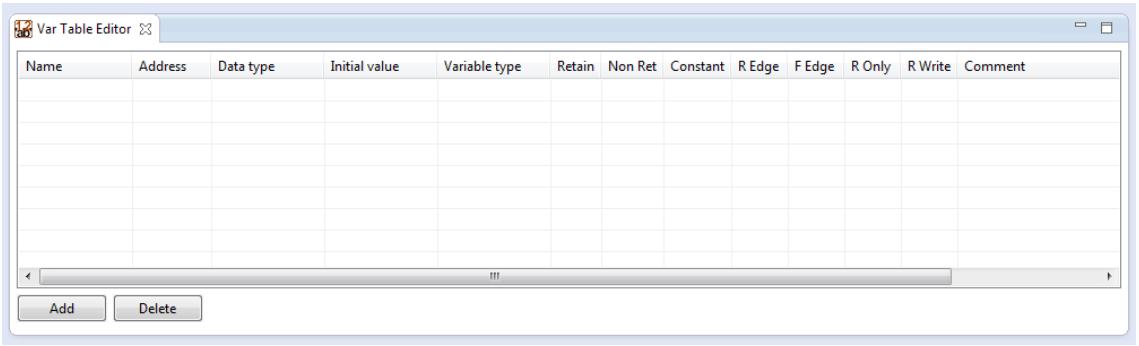

Figura 4.1 – Editor de Variáveis, módulo visual desenvolvido.

4.1.2 - Adição e remoção de variáveis através dos botões ("*Add*" e "*Delete*")

Na Figura 4.1 é possível verificar a existência de várias variáveis adicionadas recorrendo ao botão "*Add*" existente no módulo. Como se pode observar, independentemente da linha selecionada, a nova variável é inserida no final da tabela. Na Figura 4.3 verifica-se que ao clicar no botão para eliminar ("*Delete*") é eliminada a linha selecionada.

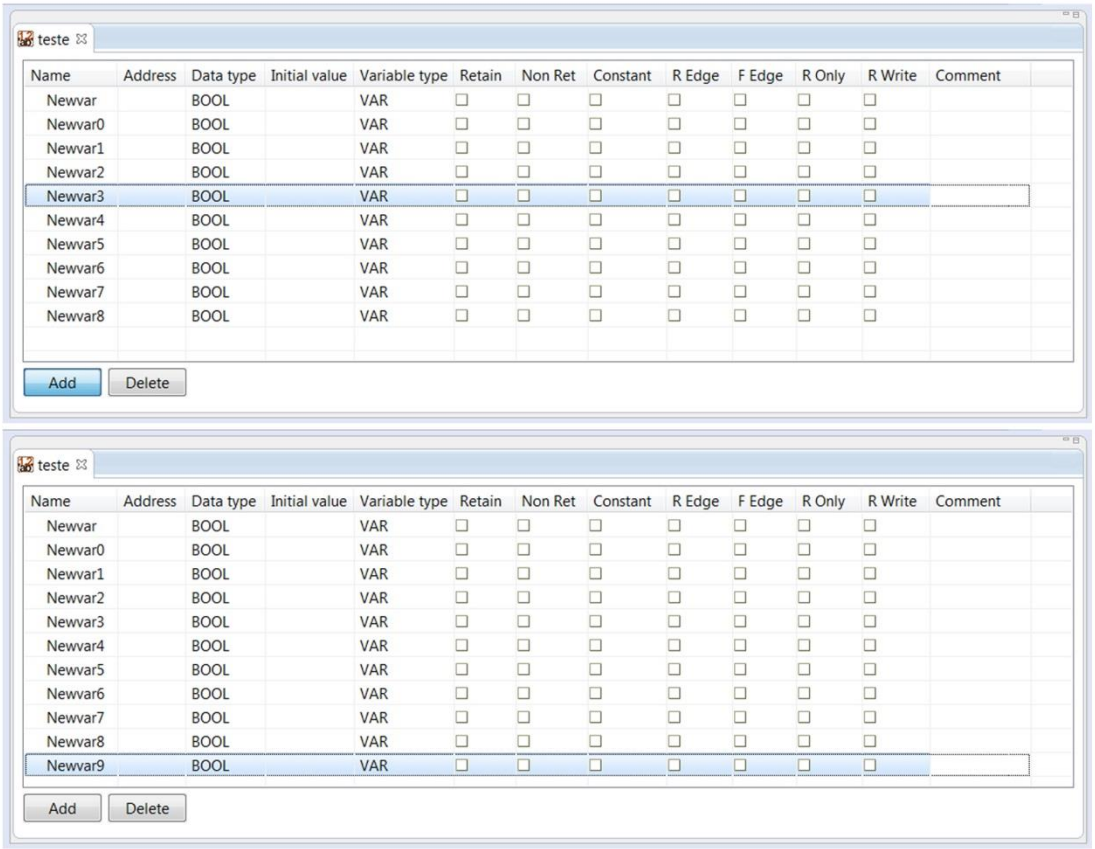

Figura 4.2 – Exemplo de adição de variável no Editor de Variáveis.

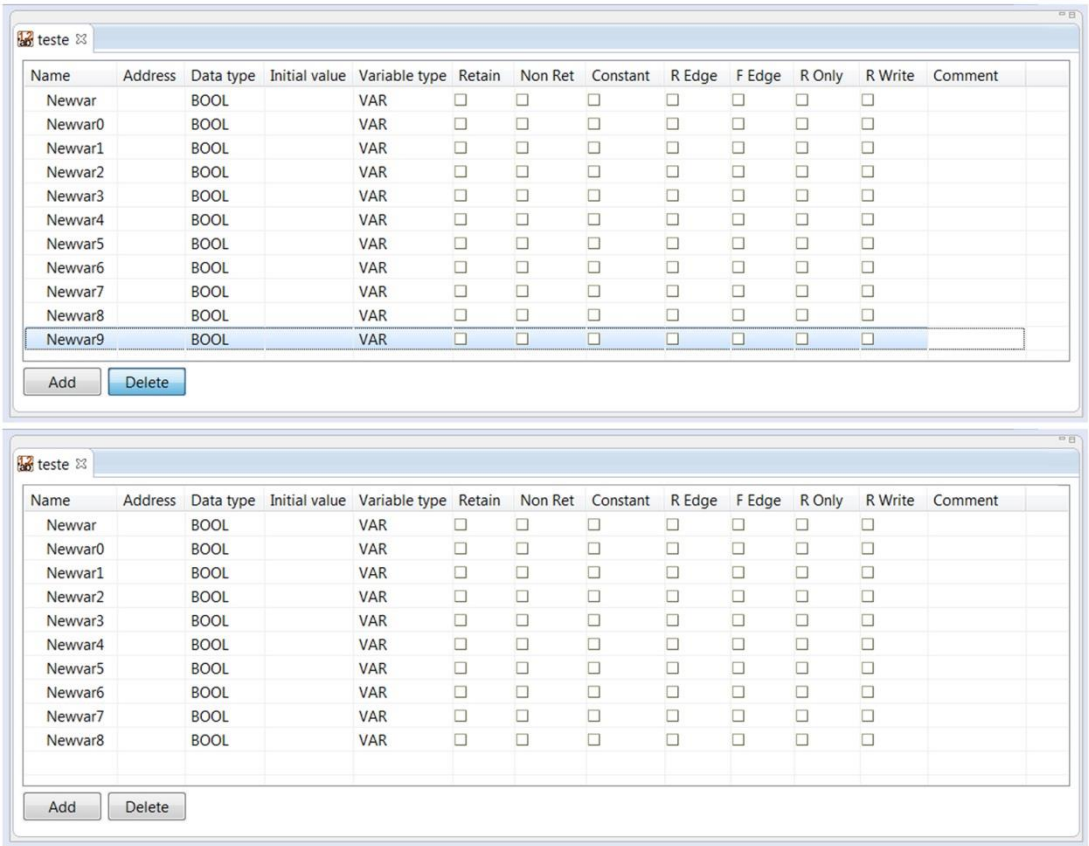

Figura 4.3 – Exemplo de remoção de variável Editor de Variáveis.

4.1.3 - Navegação pelas células e acesso à edição das células

Nesta subsecção não é apresentada imagem para transmitir os resultados da navegação pelas células, devido a uma imagem não permitir essa perceção. No entanto, foram efetuados vários testes à navegação com as teclas de direção e teclas de tabulação (com e sem a tecla "*shift*"). Tendo uma célula selecionada (sem estar em modo edição) as teclas tiveram o comportamento esperado, ou seja, a seleção deslocava-se para a direção pretendida, sendo que no caso da tabulação a célula de destino entrava em modo de edição. Com a célula em modo de edição, com as teclas de direção para a esquerda e para a direita, deslocavam o cursor na direção pretendida. Relativamente às teclas de direção para cima e para baixo, quando a edição é de apenas uma linha, estas deslocam o cursor para a esquerda e direita respetivamente, quando a edição é em células que contem edição tipo caixa de combinação, estas teclas para transição a edição para escolha superior ou inferior, respetivamente. A ação da tecla de tabulação em modo de edição insere o valor que estava a ser editado na célula e passa à célula seguinte (ou anterior caso esteja a tecla "*shift*" premida) em modo edição, em exceção se for uma célula do tipo de caixa de seleção, que apenas a seleciona.

Além destas teclas foram, também, testadas as teclas "*home*", "*end*", "*page down*" e "*page up*". Sendo que quando estava apenas selecionada a célula: a tecla "*home*" saltava para a primeira linha; a tecla "*end*" saltava para a última linha; as teclas "*page up*" e "*page down*", tem um comportamento que se pode separar em duas fases. A tecla "*page up*", numa primeira fase salta para primeira linha mostrada e depois salta na direção superior o número de células correspondentes ao número de células mostrado (ou o número mais próximo

possível). A tecla "*page down*" tem um comportamento semelhante, na direção inversa, ou seja, no sentido descendente da tabela. Com a célula em modo de edição, a tecla "*home*" e a tecla "*page up*" movem o curso para o início do texto e as teclas "*end*" e "*page down*" movem o cursor para o final do texto.

Por fim foram também testadas as teclas que permitem a entrada em modo de edição da célula selecionada, sendo elas a tecla "*enter*", "espaço", "apagar à esquerda" (*backspace*) e "F2". Foi também testada o uso das teclas "*enter*" e "*esc*" para sair do modo de edição, sendo que se confirma que o "*enter*" valida a edição efetuada e o "*esc*" funciona como cancelamento da edição.

Foi também testado a iteração com o rato. Com um clique do rato sobre a célula esta fica selecionada. Com um duplo clique a célula entre em modo de edição. Sendo que como referido no desenvolvimento, as células de edição tipo caixa de seleção, apenas altera o seu estado e ficam selecionadas, devido a não terem modo de edição.

A Figura 4.4 apresenta os modos de edição. Esta figura é uma compilação dos modos de seleção que foi feita da captura do ecrã, e foi editada de forma a realçar apenas o resultado num espaço mais pequeno. É descrito que na quarta imagem a que se encontra mais à direita é uma união entre caixa de combinação e edição textual. Como se pode verificar, existe a opção "INT", que é a escolha que está selecionada e que está inserida na célula ativa. Pode-se verificar também que na célula superior à célula ativa está inserido o texto "INT(2..56)".

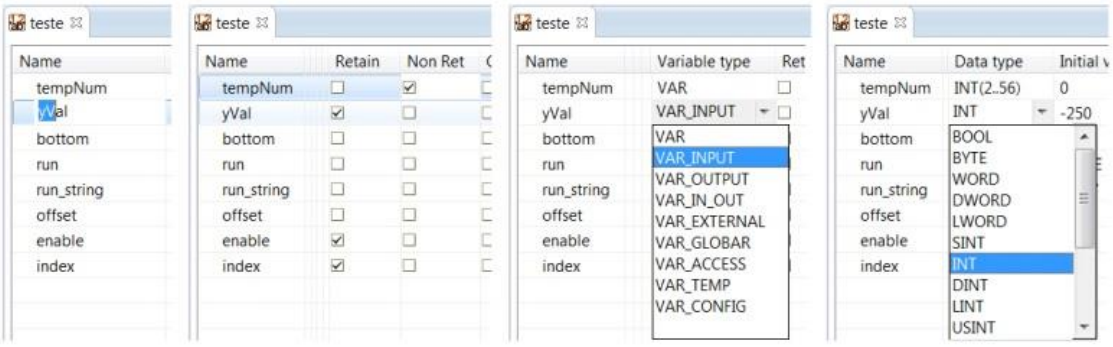

Figura 4.4 – Apresentação do resultado dos tipos de edição das células. Da esquerda para a direita pode-se verificar o tipo de edição textual, o tipo de edição de caixa de seleção, o tipo de edição de caixa combinada e, por fim, união entre caixa de combinação e edição textual.

4.1.4 - Adição e Remoção de variáveis pelo uso do teclado

Foram efetuados testes para verificar o funcionamento do conjunto das teclas que procedem à adição de novas variáveis e remoção de variáveis existentes na tabela.

A discussão destes resultados foi separada da subsecção anterior devido a estes testes terem uma intervenção diferente na tabela. Enquanto os testes anteriores, apenas alteram a célula atual selecionada, o modo de edição ou o valor da célula. Estes testes intervêm na tabela de forma a inserir uma nova variável, que pode ser colocada entre duas linhas existentes, antes de todas as linhas já criadas ou no final da tabela, assim como a eliminação de uma linha pode ser em qualquer linha. Ou seja estas ações interferem não apenas com a linha ou célula selecionada, mas com toda a tabela.

Foram efetuados testes para adição de uma nova variável na linha anterior através do conjunto de teclas "*ctrl*" + "*shift*" + "*enter*", estes testes foram realizados para inserir uma variável antes da primeira linha, estando a primeira linha selecionada, e em linhas seguintes (saltando algumas linhas entre as posições de teste) até ser inserida uma nova variável na penúltima linha, estando a última linha selecionada (Figura 4.5).

Foram igualmente efetuados testes extensivos ao longo de várias posições da tabela para a criação de uma variável na linha seguinte, com o conjunto de teclas "*ctrl*" + "*enter*". Começando com a criação de uma variável nova na segunda linha, tendo a primeira selecionada, passando por vários testes ao longo das linhas existentes e terminando com a seleção na última linha, que cria uma nova variável numa nova linha (Figura 4.6).

Em ambas as operações a inserção de uma nova variável, antes de variáveis que já existem, aumenta em uma unidade o número de variáveis existentes e move todas as variáveis seguintes uma linha para baixo.

Foram também realizados testes para a eliminação de variáveis da tabela premindo a tecla "*delete*". Seguindo a mesma metodologia, começando por eliminar a primeira linha, e ir eliminando algumas linhas até eliminar a última linha (Figura 4.7).

Nestas operações ao eliminar uma linha o número de variáveis na tabela diminui uma unidade, e todas as linhas a seguir à linha, que entretanto foi eliminada, sobem uma posição.

Estas operações apenas se verificam se a célula estiver apenas selecionada. Se a célula estiver em modo de edição, o conjunto de teclas "*ctrl*" + ("*shift*" +) "*enter*" não proporciona qualquer efeito e a tecla "*delete*" apaga o caracter que se apresente à direita (se existir) ou a seleção. Este efeito é esperado, uma vez que se pretende manter a coerência entre adicionar e remover variáveis e entre a edição de teclas e o menu de contexto do rato (subsecção seguinte).

4.1.5 - Testes à adição e remoção de variáveis pelo menu de contexto

À semelhança dos testes realizados na subsecção anterior, foram efetuados testes para verificar o funcionamento das opções do menu de contexto do rato, quando tendo uma célula selecionada. As ações das opções do menu de contexto são as mesmas ações referenciadas na subsecção anterior que procedem à adição de novas variáveis e remoção de variáveis existentes na tabela.

Estes testes estão discutidos em separado apenas devido ao facto do interface ser diferente, uso do teclado ou uso do rato. No entanto, a metodologia utilizada para os testes foram os mesmos, pelo que a descrição dos testes em baixo é algo repetitiva. Também, devido à metodologia e às ações serem as mesmas foram criadas figuras de referência de um resultado tipo usando ambas as interfaces.

Foram efetuados testes extensivos para adição de uma nova variável na linha anterior através da escolha da opção do menu de contexto "*Insert Var before*", estes testes foram realizados para inserir uma variável antes da primeira linha, estando a primeira linha selecionada, e em linhas seguintes (saltando algumas linhas entre as posições de teste) até ser inserida uma nova variável na penúltima linha, estando a última linha selecionada (Figura 4.5).

Seguindo a mesma metodologia, foram feitos testes com a escolha da opção do menu de contexto "*Insert Var after*" (Figura 4.6) e outros com a escolha da opção do menu de contexto "*Remove Var*" (Figura 4.7).

Tal como referido na subsecção anterior, em ambas as operações de inserção de uma nova variável, aumenta em uma unidade o número de variáveis existentes e move todas as variáveis seguintes ao local onde a variável será inserida uma linha para baixo. Nas operações de remoção da variável, a linha selecionada é removida, as linhas seguintes sobem uma posição e o número de variáveis na tabela diminui uma unidade

Estas opções só aparecem no menu de contexto se a célula estiver em modo de seleção. Em modo de edição, o menu de contexto é o menu padrão para edições de texto.

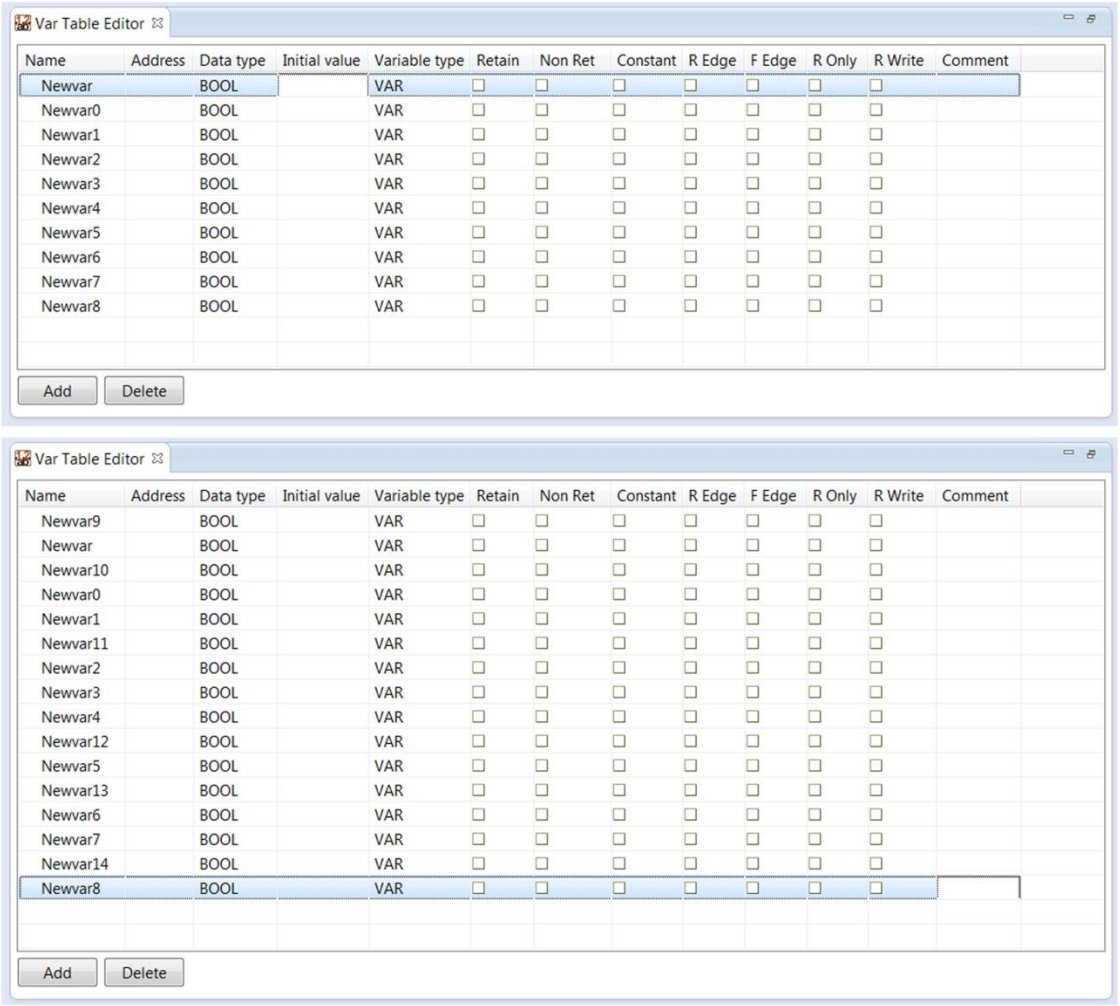

Figura 4.5 – Resultado exemplo (tabela de baixo) do uso da opção do rato "*Insert Var before*" e da combinação de teclas "*ctrl*" + "*shift*" + "*enter*" em células selecionadas. Iniciou-se na seleção da primeira de 10 variáveis (tabela de cima), e repetiu-se uma das opções em algumas linhas, inclusive na última.

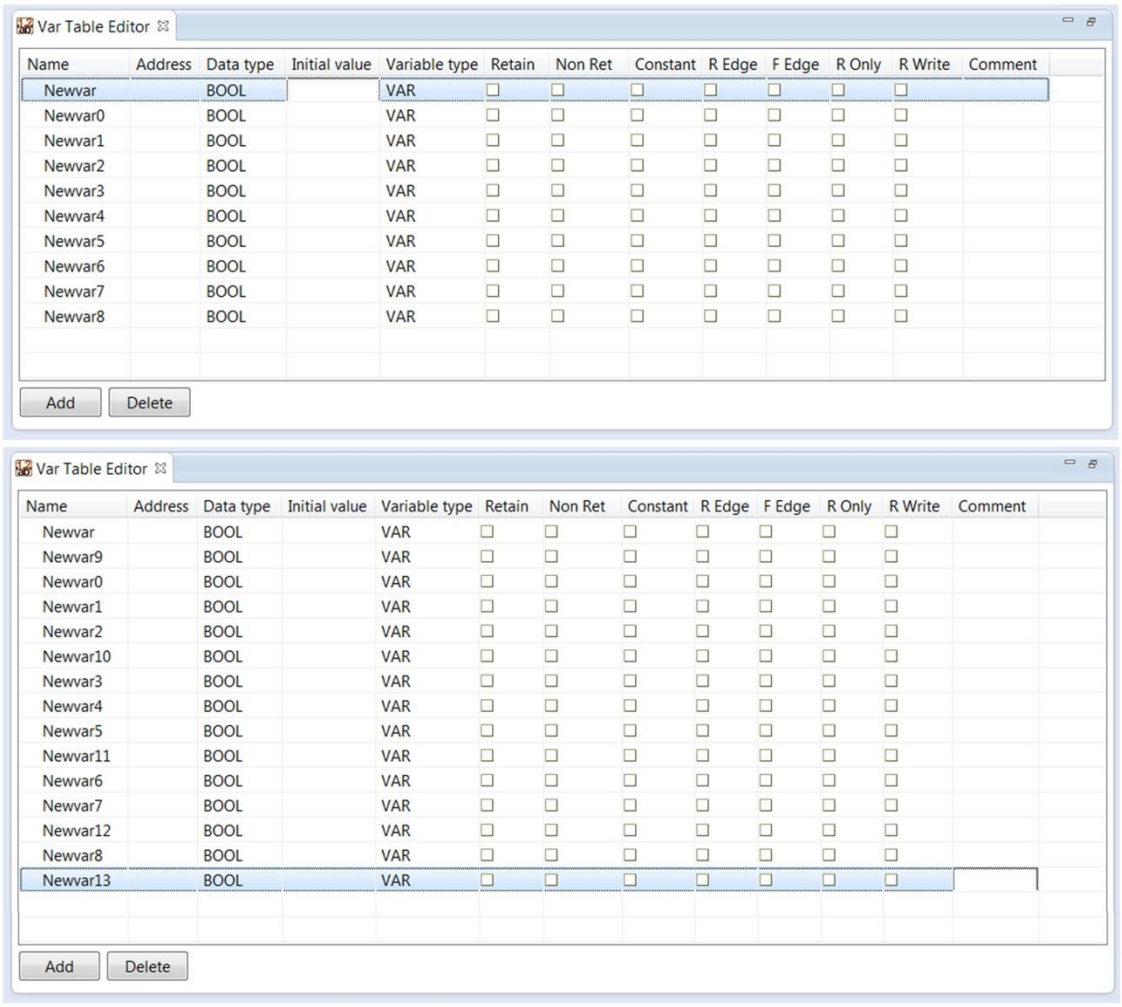

Figura 4.6 – Resultado exemplo (tabela de baixo) do uso da opção do rato "*Insert Var after*" e da combinação de teclas "*ctrl*" + "*enter*" em células selecionadas. Iniciou-se na seleção da primeira de 10 variáveis (tabela de cima), e repetiu-se uma das opções em algumas linhas, inclusive na última.

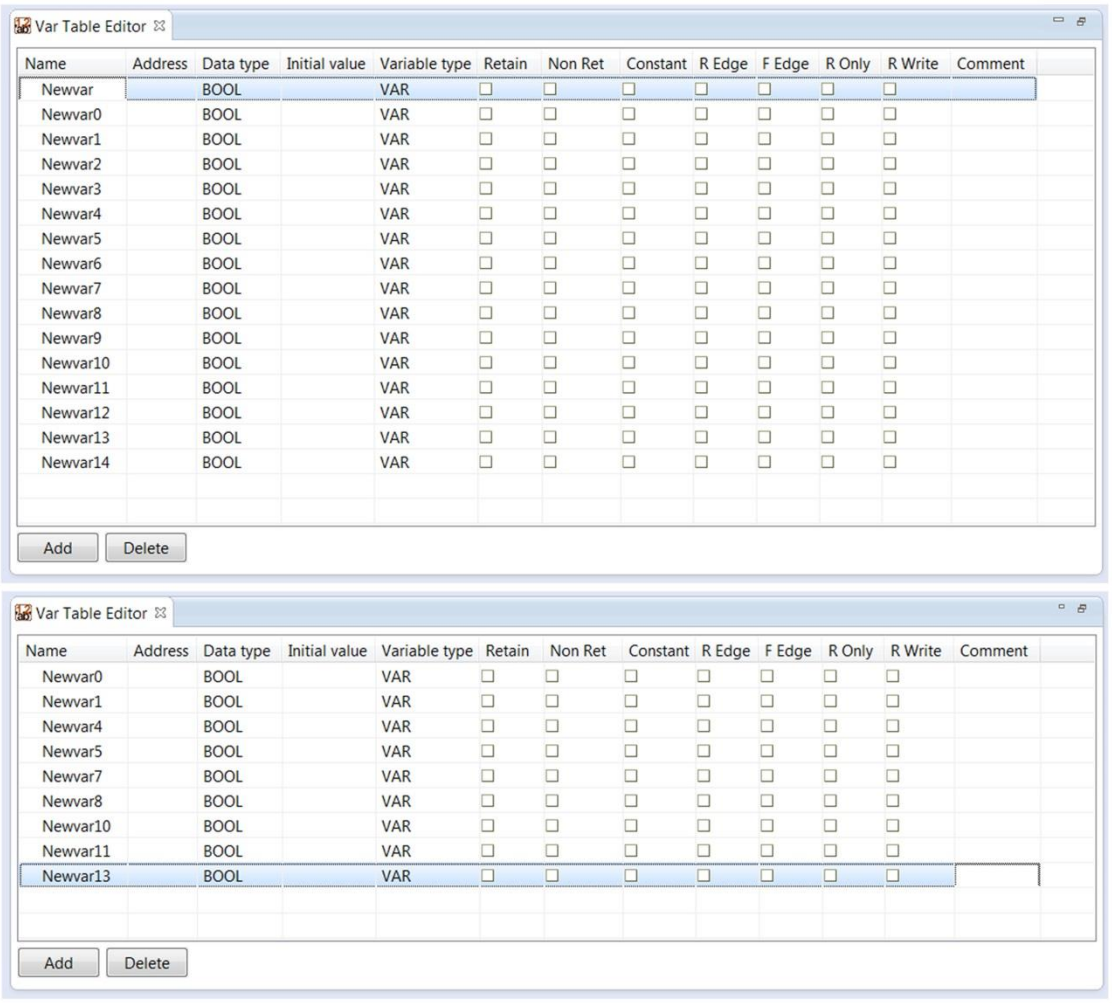

Figura 4.7 – Resultado exemplo (tabela de baixo) do uso da opção do rato "*Remove Var*" e da tecla "*delete*" em células selecionadas. Iniciou-se na seleção da primeira de 16 variáveis (tabela de cima), e repetiu-se uma das opções em algumas linhas, inclusive na última.

4.1.6 - Ficheiros "*.tvar*"

Para a realização de testes à gravação de ficheiros e leitura das variáveis, foi necessário primeiro criar ficheiros e abrir esses ficheiros através do navegador de projetos do Eclipse. Uma vez que os estes testes foram realizados antes do desenvolvimento do Navegador para apresentar e cria esse tipo de ficheiros.

Esses ficheiros eram criados vazios apenas com o nome e extensão. Depois eram abertos e era visível o nome do ficheiro na descrição do separador, respeitante a cada ficheiro, ao invés de estar designação "Var Table Editor". Quando aberto pela primeira vez o ficheiro não continha nenhuma variável. Criavam-se variáveis e fechava-se o ficheiro. Depois para perceber se a informação era guardada era confirmado pelo tamanho do ficheiro se tinha informação. Por fim, voltava-se a abrir o ficheiro no editor para verificar se este continha as variáveis anteriormente criadas. Depois eliminava-se e criava-se novas variáveis e procedia-se à verificação do ficheiro e à sua abertura. Esta última repetição no teste era para confirmar que o ficheiro não gravava apenas com o ficheiro vazio, mas que também as alterações eram guardadas. Na Figura 4.8 pode ver-se os passos e um exemplo destes testes.

Figura 4.8 – Resultado de teste a um ficheiro do tipo “*.tvar*” com 8 passos.

4.1.7 - Criação de 2500 variáveis numa tabela

Pretendia-se realizar testes com uma tabela considerada de grandes dimensões para o objetivo do projeto. Para esses testes, foi decidido criar tabelas com 2500 variáveis para testar o comportamento de alterações, adição de novas variáveis e eliminações de variáveis. Sendo que esses testes seriam feitos ao longo da tabela. E também testar os tempos de abertura. Para testar a escolha feita na decisão de criar uma tabela que ativa todos as linhas na abertura, ao invés de ir obtendo os valores à medida que se vai deslocando na tabela. A Figura 4.9 apresenta um desses ficheiros.

Dos testes pode-se referir que como era esperado a tabela de grandes dimensões demora mais tempo a abrir. Não foi usado nenhum tipo de equipamento de medição de tempo, nem nenhum programa que detetasse os tempos de abertura com precisão, porque foi decidido que a diferença e tempo de abertura não são de grandeza suficiente para requerer esse tipo de medição. Em termos comparativos, é de comparar a sensação de menos de 1 segundo, quase imediata, a abrir uma tabela com um número reduzido de linhas e um tempo de espera

por volta de 3 segundos para uma tabela de pouco de mais 2500 variáveis. Ao nível de adicionar e eliminar variáveis quer no início, a meio ou no final da tabela, não se nota atraso que denote observação. Na edição das células textuais nota-se um ligeiro atraso na validação da informação. Sendo que esse atraso é ligeiramente maior na edição das caixas de seleção, que são as células que apresentam imagens.

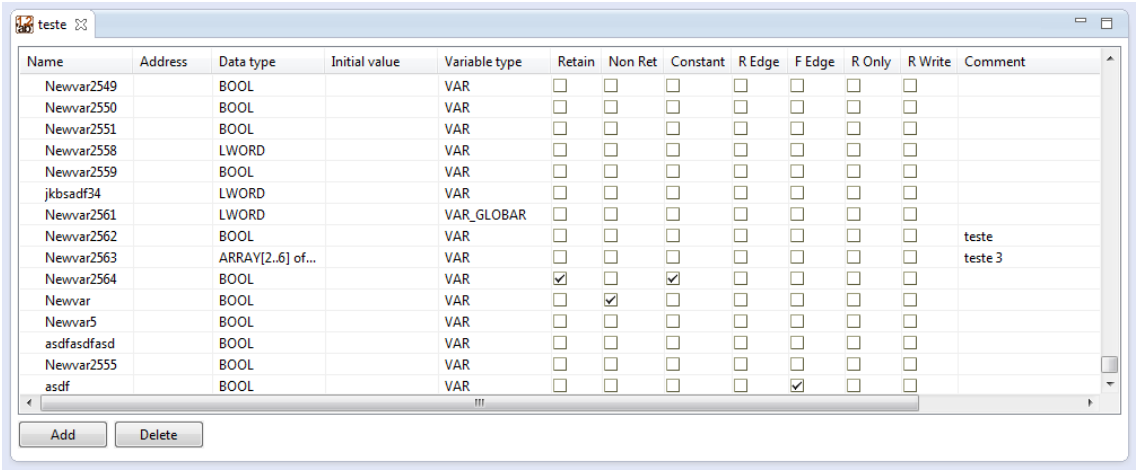

Figura 4.9 – Editor de Variáveis, ficheiro exemplo de teste com mais de 2500 variáveis.

4.1.8 - 20 ficheiros abertos

Foram realizados testes de performance com cerca de 20 ficheiros do tipo "*tvar*" abertos, juntamente com os ficheiros dos editores e tendo também ficheiros de 2500 variáveis.

Na Figura 4.10 pode-se verificar um desses testes, em que é possível ver um dos ficheiros de mais de 2500 variáveis aberto.

Dos testes efetuados pode-se referir que os tempos de resposta são praticamente os mesmos com 1 ou com os 20 ficheiros abertos, mesmo para os ficheiros de grandes dimensões não é notado um atraso pronunciado na abertura ou edição dos mesmos. No entanto, como o aumento do número de ficheiros ativos o tamanho dos separadores diminui o que provoca uma falha nas leituras dos nomes dos ficheiros.

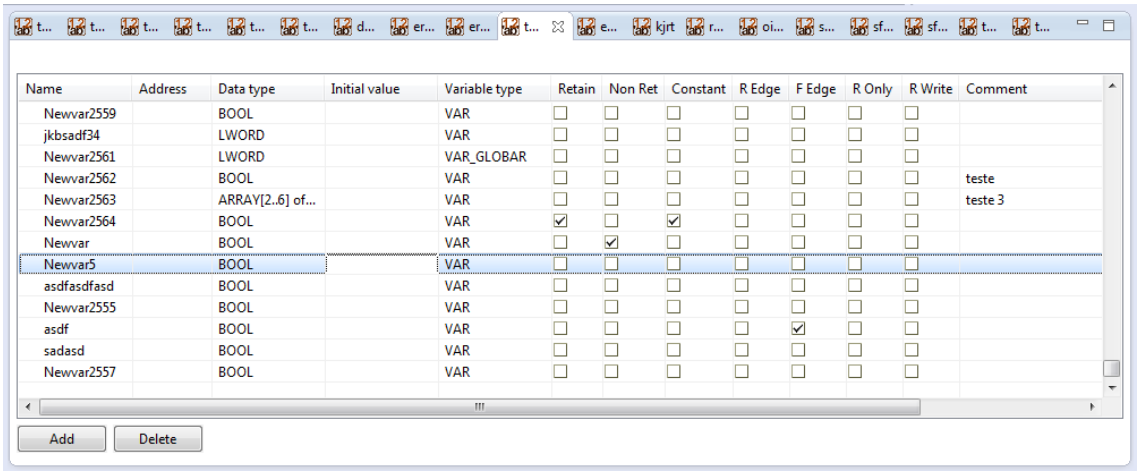

Figura 4.10 – Resultado do teste de múltiplos tipos de ficheiros abertos.

4.2 - Navegador

Apresenta-se nesta secção os resultados relativos aos testes efetuados ao desenvolvimento efetuado ao Navegador. Para garantir uma melhor perceção dos resultados obtidos estes são apresentados com figuras. Estas figuras podem ser simples captura de ecrã ou imagens editadas com o intuito de salientar o resultado obtido.

4.2.1 - Visualização de projetos fechados e de projetos de outra natureza

No desenvolvimento do Navegador uma das melhorias realizadas no *plug-in* foi a apresentação dos projetos que se encontravam fechados e a apresentação dos ficheiros de outras naturezas. Sendo que os projetos da natureza IEC61131-3 possuiriam um decorador, para salientar a sua natureza, e seriam os únicos que iriam apresentar ramificação.

Como se pode verificar na Figura 4.11, os projetos "teste3" e "teste4" encontram-se fechados e todos os restantes projetos se encontram abertos, informação dada pelo ícone. Dos projetos abertos, todos, à exceção do "MyMarker", apresentam o ***Decorator*** e apresentam ramificação. Isto acontece, porque à exceção do "MyMarker" todos os restantes são da natureza IEC61131-3. A ramificação é visível pela seta atrás do ícone, que está negra e inclinada para baixo do ícone quando a ramificação se encontra expandida, caso contrário encontrar-se branca e a apontar para o ícone.

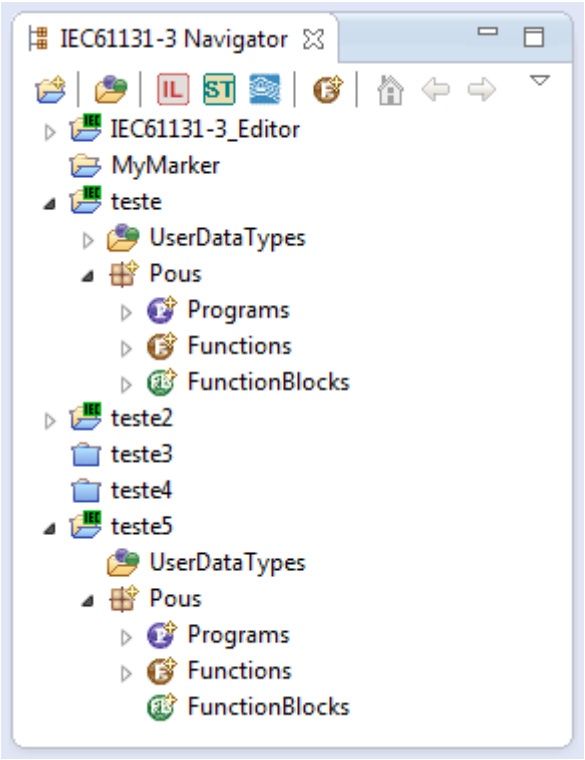

Figura 4.11 – Navegador, apresentação do estado do projeto e natureza quando aberto.

4.2.2 - Visualização de ficheiros "*.tvar*"

Com a criação de ficheiros "*.tvar*" foi necessário alterar o navegador para apresentar estes ficheiros e para proceder à abertura do modulo correspondente com o ficheiro atribuído aquando do duplo clique ou da ativação do ficheiro com a pressão da tecla "*enter*".

Na Figura 4.12 é apresentada uma ramificação de um projeto onde se pode visualizar a presença de vários ficheiros do tipo "*tvar*".

Apesar de não ser apresentada uma figura a descrever os passos de abertura de um módulo, estes testes foram realizados tendo o resultado sido sempre a abertura do módulo com a respetiva descrição referente ao nome do ficheiro no separador.

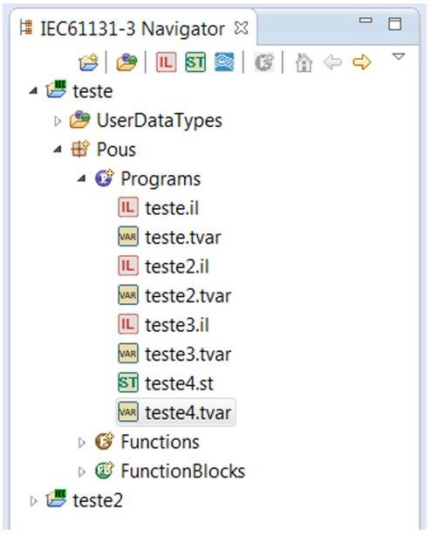

Figura 4.12 – Visualização dos ficheiros do tipo "*.tvar*".

4.2.3 - **POUelement**

Na Figura 4.13 é possível ver o Navegador antes e depois de terem sido inseridos os **POUelements**, onde se verifica que os elementos de um POU são agrupados no **POUelement**.

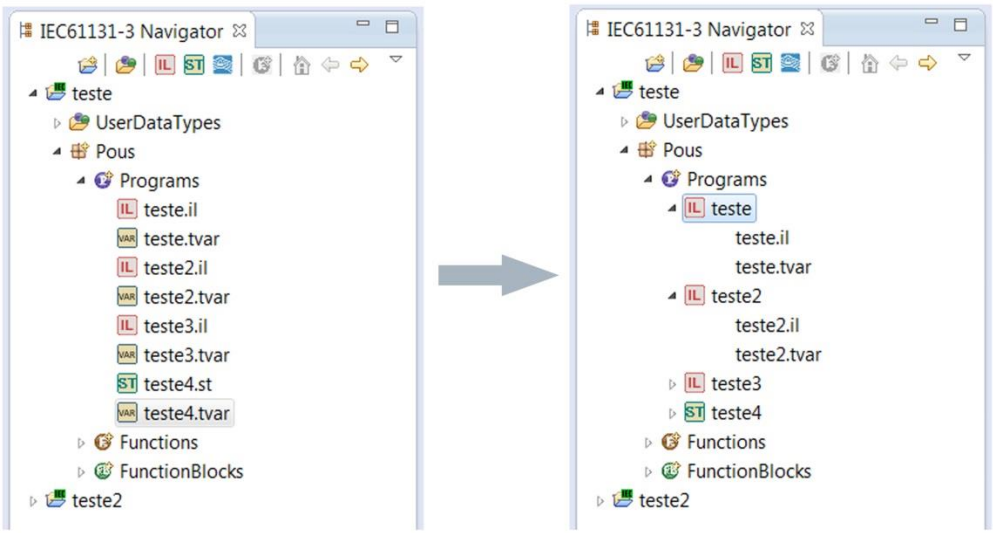

Figura 4.13 – Visualização do **Navegador** antes e depois dos **POUelements** e seus elementos.

Quando se efetua um duplo clique sobre um **POUelement**, ou quando este está selecionado e se pressiona a tecla "*enter*", são abertos ambos os elementos do **POUelement**. O esquema da Figura 4.14 é apresentado esse processo de abrir os elementos de um **POUelement**. Que foi o resultado obtido nos testes efetuados.

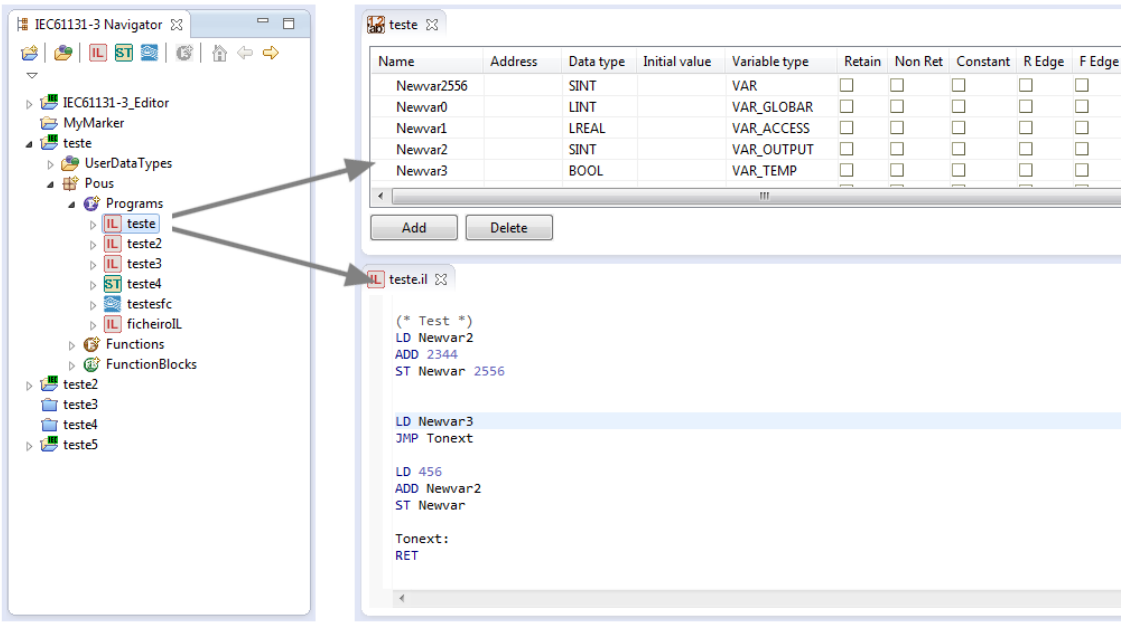

Figura 4.14 – Esquema de abertura de ambos os elementos do POUelement com um duplo clique.

4.2.4 - Criação dos POU

No desenvolvimento do **Navegador** foram efetuado também melhorias para uma criação mais direta dos POU (Funções, Blocos de Funções e Programas), independentemente do seu tipo de linguagem, IL, ST ou SFC. Foram criadas ferramentas para obter na árvore do **Navegador** a localização direta da diretoria e solicitando apenas o nome do ficheiro, criar um ficheiro com a extensão correspondente à linguagem pretendida, na pasta pretendida. Agora a criação de um ficheiro de uma linguagem, cria também o ficheiro "*.tvar*" para a possível criação de variáveis desse POU, e cria na árvore de navegação um **POUelement** que agrupa esses ficheiros.

Na Figura 4.15 é possível ver os passos para criação de um ficheiro do tipo de linguagem IL. Na última janela é possível verificar que foram criados ambos os ficheiros da linguagem e da criação de variáveis. O mesmo se pode verificar na Figura 4.16, relativamente à linguagem ST, que também representa uma linguagem textual. A criação de ficheiros da linguagem gráfica SFC é possível ver na Figura 4.17, que se tornou simples como as restantes linguagens, e verifica-se também que os ficheiros agora criados possuem a extensão "*.sfc*", que foi outras das melhorias do desenvolvimento desta dissertação nos projetos anteriormente desenvolvidos.

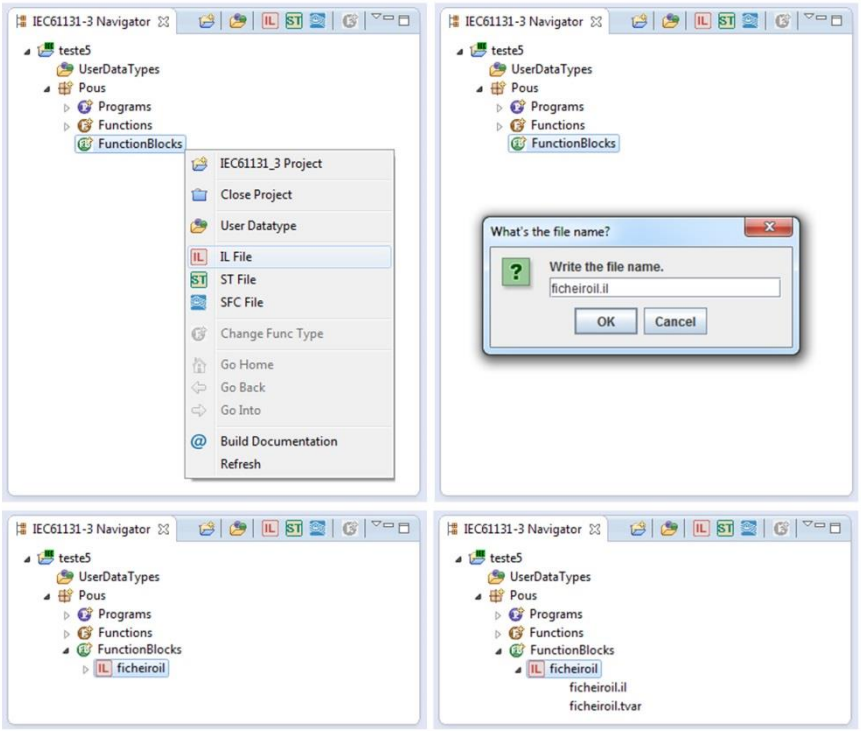

Figura 4.15 – Esquema que representa os passos de criação de um POU da linguagem IL.

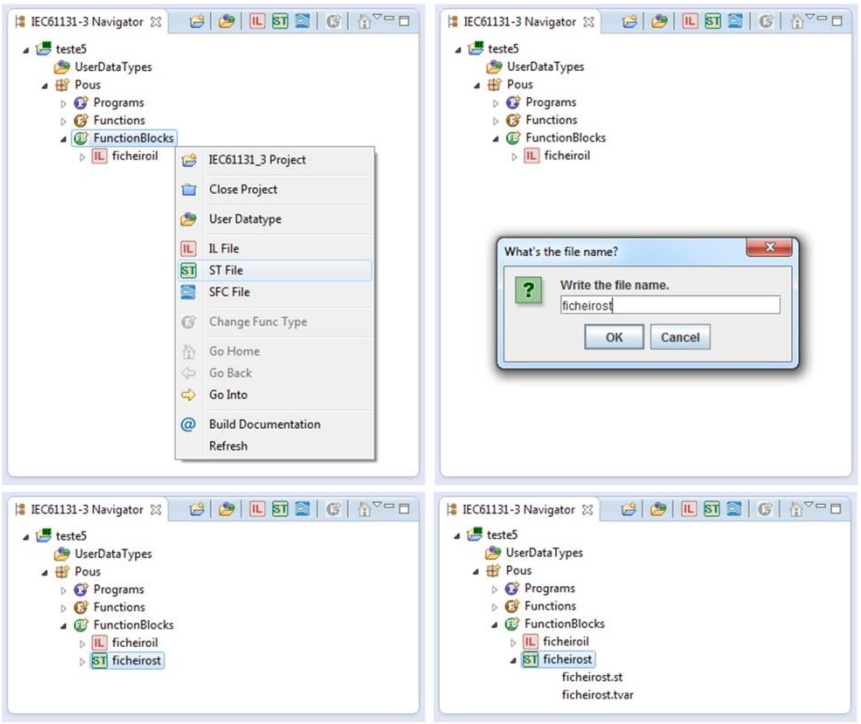

Figura 4.16 – Esquema que representa os passos de criação de um POU da linguagem ST.

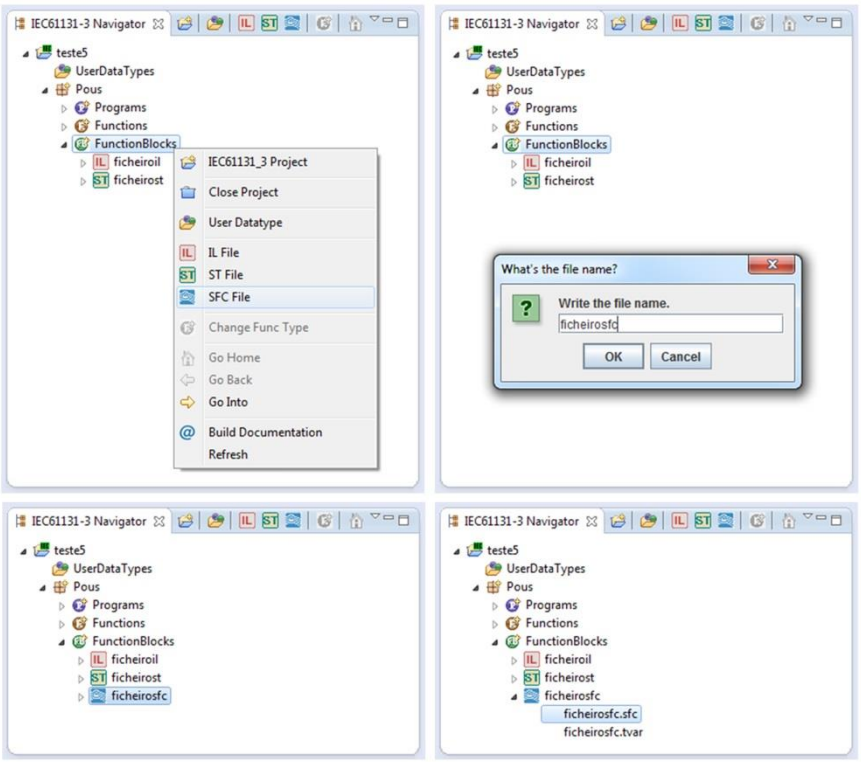

Figura 4.17 – Esquema que representa os passos de criação de um POU da linguagem SFC.

4.2.5 - Dinamização das ações e alternância entre a ação *open* e *close Project*

Outro dos desenvolvimentos realizados foi a dinamização das ações já existentes. Na Figura 4.18 pode-se verificar o resultado do desenvolvimento dessa dinamização. As duas primeiras opções "*IEC61131_3 Project*" e a "*Close Project*" não sofrem dinamização, pois é sempre possível criar um projeto e neste caso a seleção é sempre dentro de um projeto aberto e que, por isso, pode ser fechado. Porém verifica-se que apenas quando a seleção está no interior da pasta do projeto a opção "*User Datatype*" fica ativa. As opções para a criação de ficheiros do tipo POU ("*IL File*", "*ST File*" e "*SFC File*"), apenas ficam ativas quando a seleção é feita dentro da ramificação dos POU, elemento "*Pous*".

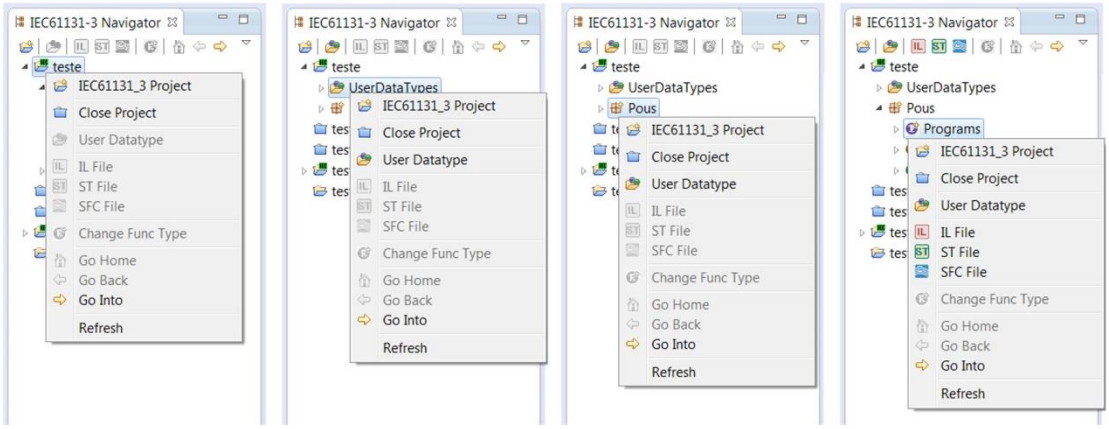

Figura 4.18 – Visualização da dinamização das opções do menu de contexto no **Navegador**.

Devido ao facto de um projeto poder apenas ter um estado de aberto ou fechado, e para não estar sempre com uma opção desativada no menu de contexto, foi decidido que as opções de abrir um projeto (“*open project*”) e de fechar um projeto (“*close project*”) apareceriam no menu de contexto em alternância dependendo do estado do projeto selecionado. Na Figura 4.19 é possível verificar o resultado desse desenvolvimento, com a alternância dessa opção dependendo se esta selecionado um projeto que está aberto ou fechado.

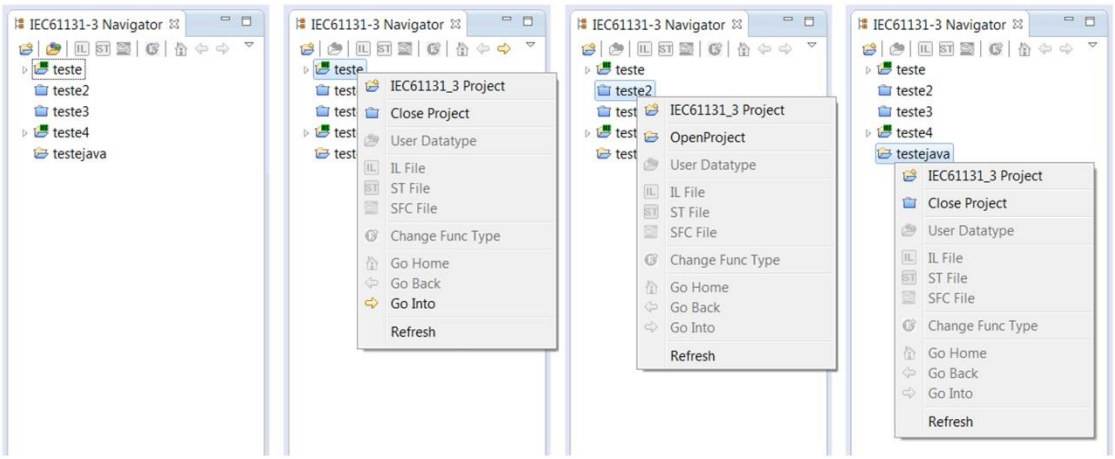

Figura 4.19 – Visualização da alternância das opções de abrir e fechar o projeto.

4.2.6 - Tipo de retorno da função (Function return type)

Foi definido que o tipo de retorno da função seria identificado no **POUelement** de cada função. A Figura 4.20 apresenta o resultado desse desenvolvimento.

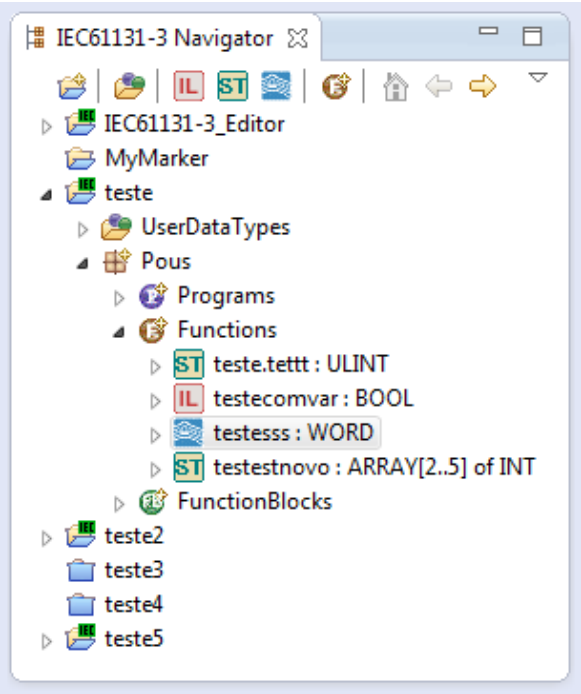

Figura 4.20 – Identificação do tipo de retorno da função no **POUelement**.

4.2.7 - Alteração do tipo de retorno da função

Na Figura 4.21 é apresentado um esquema de passos que representa a ação de modificar o tipo de retorno de uma função. Neste exemplo, procede-se à alteração do retorno do tipo "Word" para "DWord" da função "testesss".

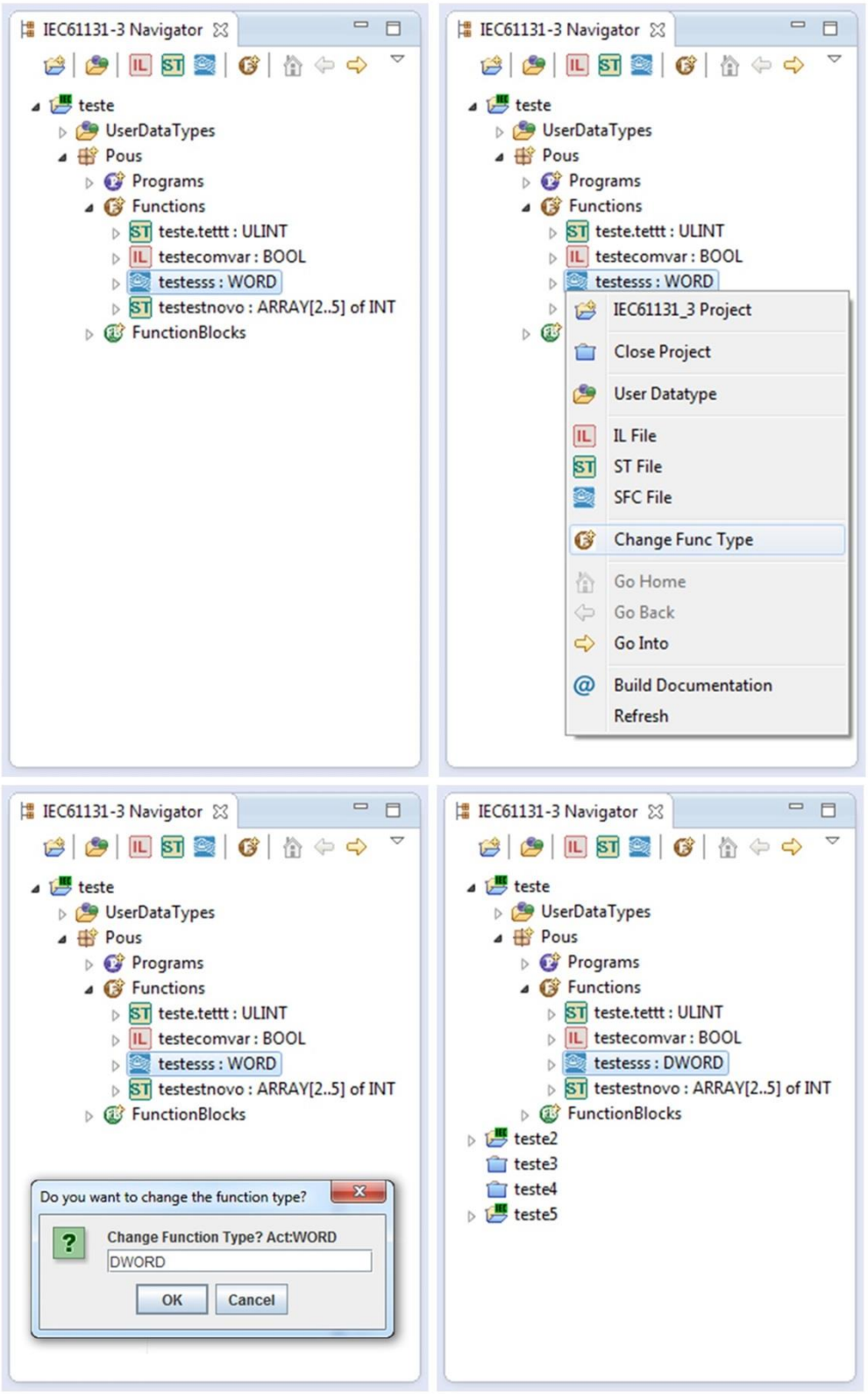

Figura 4.21 – Esquema que representa os passos para alteração do tipo de retorno de uma função.

4.3 - Editor de UDT

Nesta secção estão apresentados os resultados relativos aos testes efetuados ao Editor de UDT. Devido ao Editor de UDT ter sido desenvolvido do Editor de Variáveis, apesar de terem sido realizados testes à navegação e adição e remoção de UDT da tabela, os resultados desses testes não foram incluídos nesta dissertação porque os resultados são exatamente iguais aos resultados do Editor de Variáveis. Foi decidido por isso, dar mais ênfase aos testes que poderiam ter resultados diferentes, quer por ser uma tabela com menos colunas, quer por ser uma tabela com elementos mais complexos.

Os resultados serão acompanhados por figuras, sempre que estas transmitam uma melhor perceção dos resultados. Estas figuras podem ser simples captura de ecrã ou imagens editadas com o intuito de salientar o resultado obtido.

4.3.1 - Criação do módulo visual

Tal como no Editor de Variáveis, é apresentado primeiramente o módulo Editor de UDTs desenvolvido onde se verifica: a identificação do módulo na aba separadora; a existência de um interface do tipo tabela, sem elementos; as quatro colunas e a sua descrição; e a existência de dois botões para adicionar e remover.

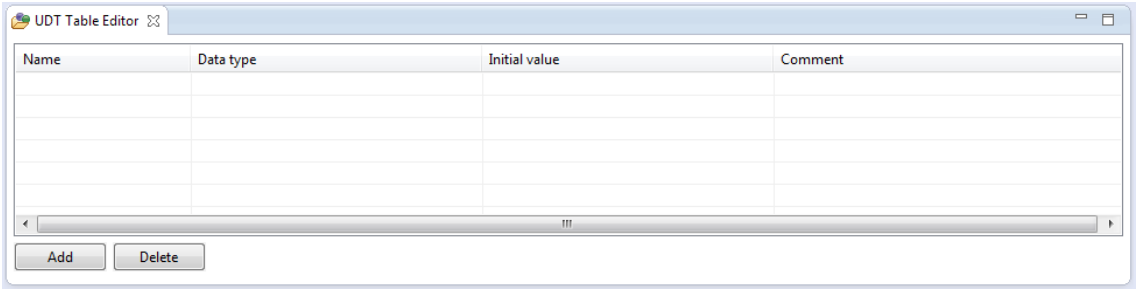

Figura 4.22 – Editor de UDT, módulo visual desenvolvido.

4.3.2 - Ficheiros "*.tudt*"

A realização dos testes, de criação de ficheiros e de gravação da informação, foi realizada recorrendo ao **Navegador**. Ao contrário dos testes realizados aos ficheiros "*.tvar*", na realização destes testes a opção de criar ficheiros "*.tdut*" e de os abrir já havia sido incluída no Navegador**.** Assim, os ficheiros do tipo "*.tudt*" foram criados e abertos a partir da árvore de navegação. Os ficheiros abertos apresentaram-se vazios e era visível o nome do ficheiro na descrição do separador, ao invés de estar designação "UDT Table Editor". Procedia-se depois à criação de UDT e fechava-se o ficheiro. Seguidamente, voltavam-se a abrir os ficheiros no editor para verificar se este continha os UDT anteriormente criados e procedia-se à de novo à alteração da informação (edição, eliminação e criação de UDT) e voltavam-se a fechar os ficheiros. Por fim, voltava-se a abrir os ficheiros para confirmar a se a última informação inserida estava guardada. Na Figura 4.23 podem ver-se os passos e um exemplo destes testes.

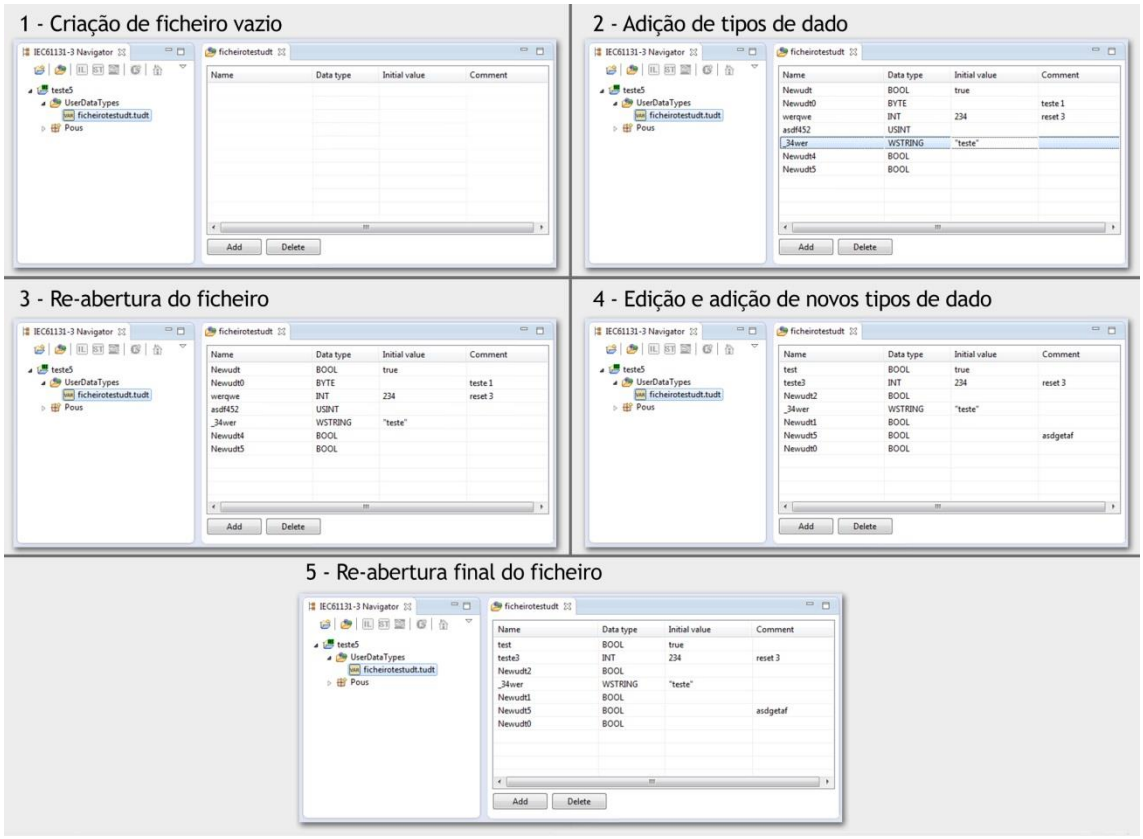

Figura 4.23 – Resultado de teste a um ficheiro do tipo "*.tudt*" com 5 passos.

4.3.3 - Criação de 2500 UDT numa tabela

Nesta subsecção são apresentado os resultados realizados à criação de uma tabela de UDT com mais 2500 UDT. Foi decidido apresentar estes resultados, mesmo já o tendo sido realizado para o Editor de Variáveis, uma vez que esta tabela tem apenas 4 colunas e não possui células com imagens (como as células do tipo de edição caixa de seleção).

Em termos comparativos de tempo de resposta entre um ficheiro UDT de grandes dimensões e um ficheiro UDT de poucas linhas, à semelhança do observado no Editor de Variáveis, o tempo de resposta de adição e remoção de linhas aparenta ser o mesmo. Relativamente à validação do valor da célula, denota-se um tempo de resposta um pouco maior. O tempo de abertura do ficheiro é de cerca de 2 segundos.

Em termos comparativos entre um ficheiro de grandes dimensões do Editor de Variáveis e do Editor de UDT, verifica-se que o tempo de resposta do Editor de UDT é menor que o tempo de resposta do Editor de Variáveis.

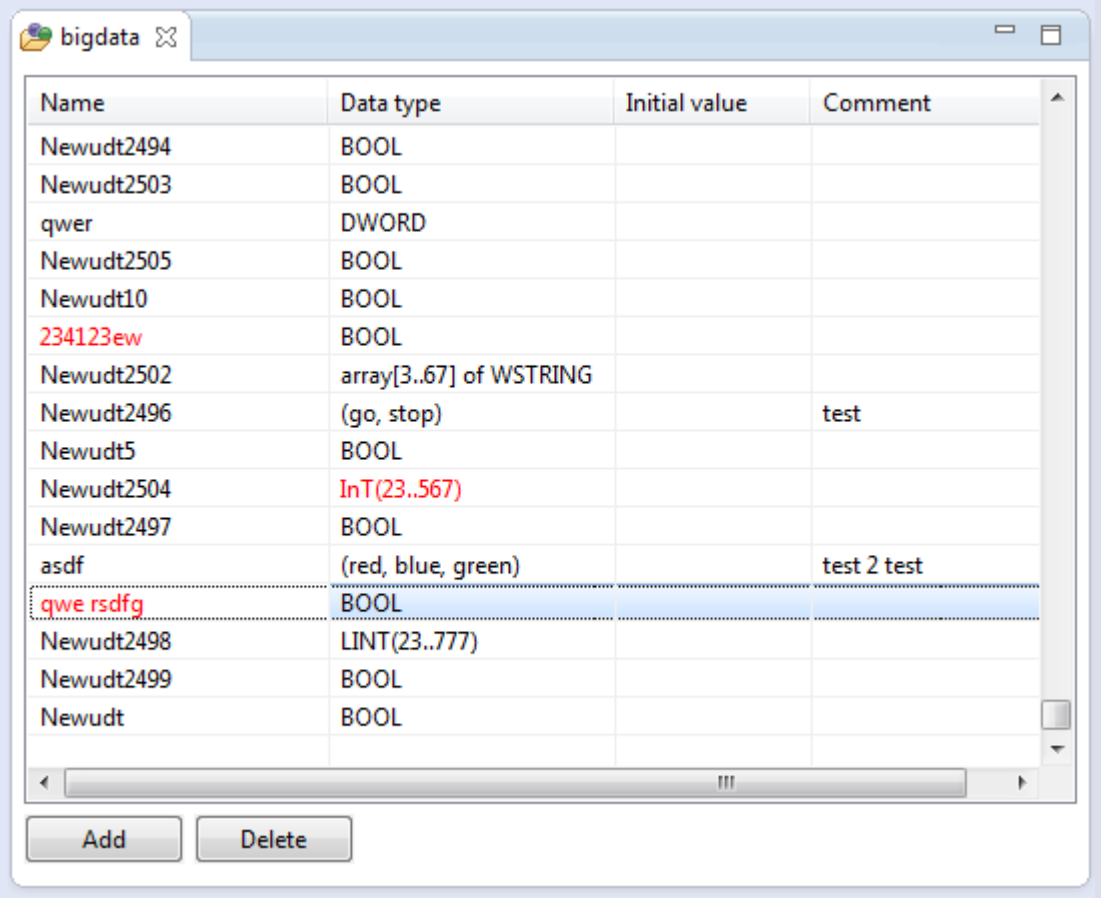

Figura 4.24 – Editor de UDT, ficheiro exemplo de teste com mais de 2500 UDT.

4.3.4 - Validação de nomes e de tipos de dados

Foram desenvolvidas tecnologias para a identificação e validação se os valores textuais inseridos na tabela são tipos elementares ou algum dos tipos de dado inseridos de acordo com a norma. Essas ferramentas foram aplicadas à coluna nome, para validação se o nome é um identificador, e à coluna tipo de dado, para identificar de qual o tipo de dado inserido e se o valor inserido respeita a norma e as regras de semântica e sintaxe do tipo de dado. Para validar estas ferramentas foram realizados testes extensivos, onde se inseriram vários tipos de valores textuais em ambas as colunas, com valores corretos e valores errados para verificar se estes eram validados. Sendo que, quando um valor textual não era validado a cor do seu texto era alterada para vermelho. Na Figura 4.25 pode-se verificar a introdução de vários valores textuais sendo que alguns estão apresentados a vermelho por serem inválidos. No campo destinado ao comentário está a justificação de porque o valor não foi validado ou a identificação de porque um valor é validado (em alguns casos).

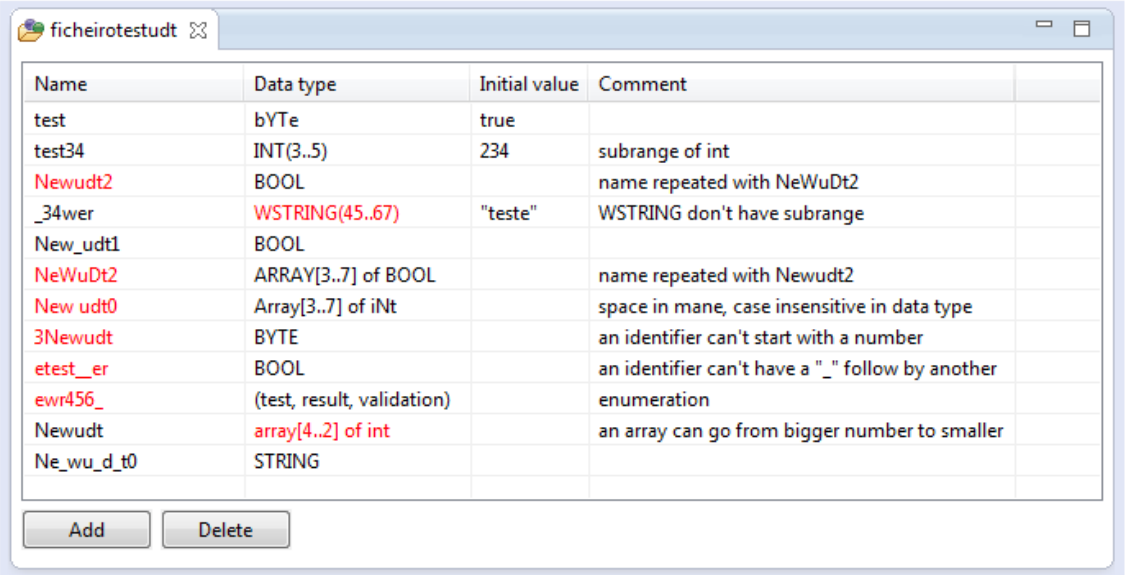

Figura 4.25 – Editor de UDT, validação dos nomes e tipos de dado.

4.4 - Reestruturação do código

Nesta secção serão apresentados os resultados obtidos dos testes realizados ao desenvolvimento efetuado relativamente à secção 3.4, onde foi realizada a restruturação do código do projeto e foram criadas ferramentas de interligação dos *plug-ins*.

Não existe uma secção com resultados dos testes da reestruturação ou alteração do código realizada, existe sim, a apresentação de resultados de ferramentas que foram criadas dessa restruturação.

Assim na primeira subsecção será apresentado um resultado tipo da implementação da classe **TreeIECProject** e na segunda subsecção será apresentado um resultado tipo da implementação dos ***Observers Pattern***.

4.4.1 - TreeIECProject: TreeParent + UDT

Nesta subsecção será apresentado um resultado tipo da implementação do **TreeIECProject**. Para se perceber melhor o resultado obtido é de referir que a implementação **TreeIECProject** tem por objetivo agrupar na árvore de navegação a informação existente dos UDT criados. Para que quando declarada uma variável com um desses tipos de dado o **Editor de Variáveis** aceda a essa informação e valide o tipo de dado inserido. Por exemplo, na Figura 4.26, pode ver-se que os tipos de dado “semáforo” e “permissao” são validados pelo **Editor de Variáveis**, devido a terem sido criados no Editor de UDT, mas o tipo de dados “naocriado” não é validado, pois não existe.

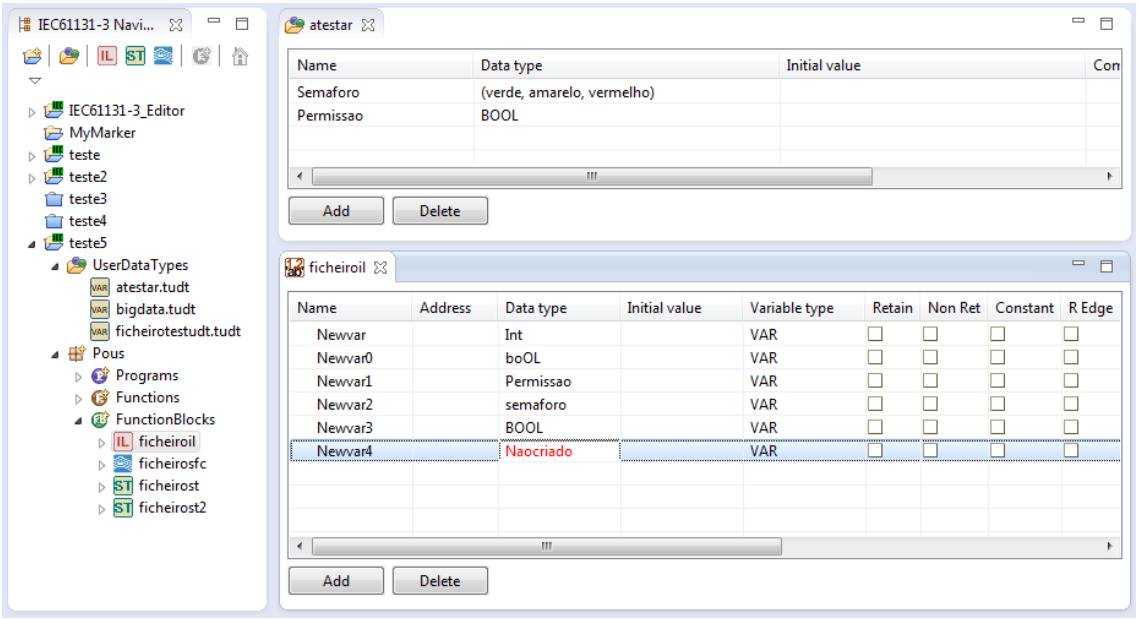

Figura 4.26 – Resultado de um teste exemplo à implementação **TreeIECProject** (para visualização de ambos os editores, o Editor de UDT foi colocado em cima e o Editor de Variáveis foi colocado em baixo).

4.4.2 - Observers

Nesta subsecção será apresentado um resultado tipo da implementação de um ***Observer***. Na Figura 4.27, que se apresenta dividida em 3, é apresentado um esquema de imagens sequenciais que tem como objetivo apresentar o resultado de um ***Observer***. Para uma melhor perceção, como referido um ***Observer*** funciona como uma subscrição de atualizações, no exemplo o **Editor de Variáveis** tem subscrito as atualizações da **árvore de navegação**, pelo que quando esta é atualizada, alerta o editor dessa atualização. Por sua vez o editor trata de utilizar essa informação atualizada sem que o utilizador tenha de efetuar alguma operação.

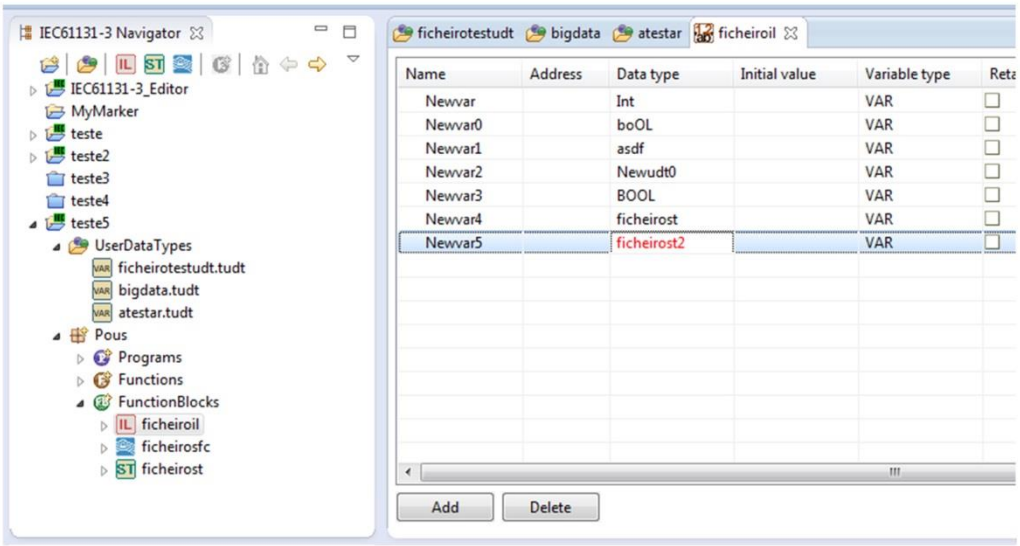

Figura 4.27 – Esquema para demonstrar o resultado da implementação de ***Observer patterns***.
Erro do tipo de dado que representa uma instância de um bloco funcional por este não existir (parte 1 de 3).

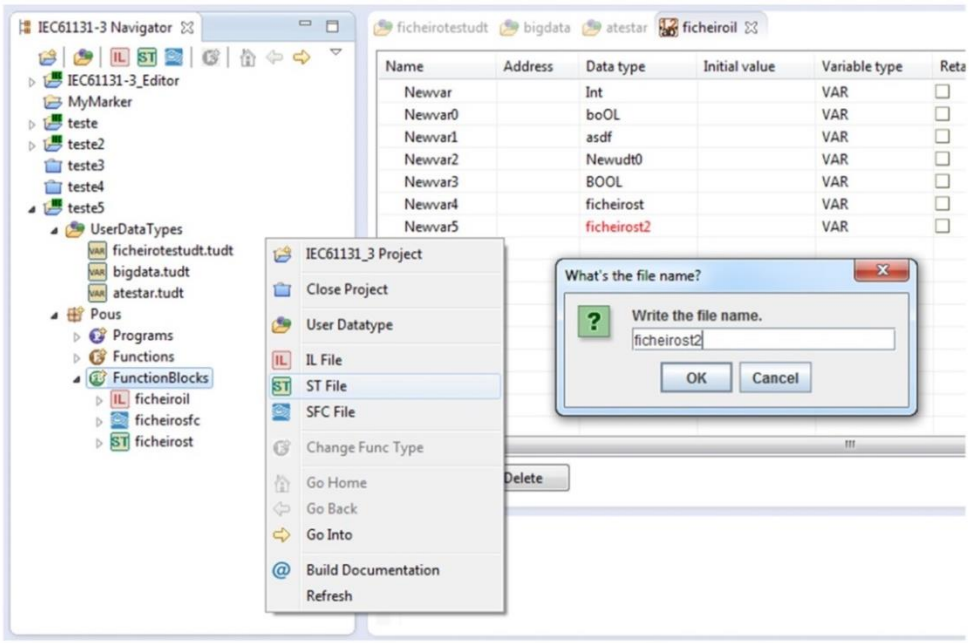

Figura 4.27 – Esquema para demonstrar o resultado da implementação de ***Observer patterns***. Criação de bloco funcional em falta "ficheirost2" (parte 2 de 3).

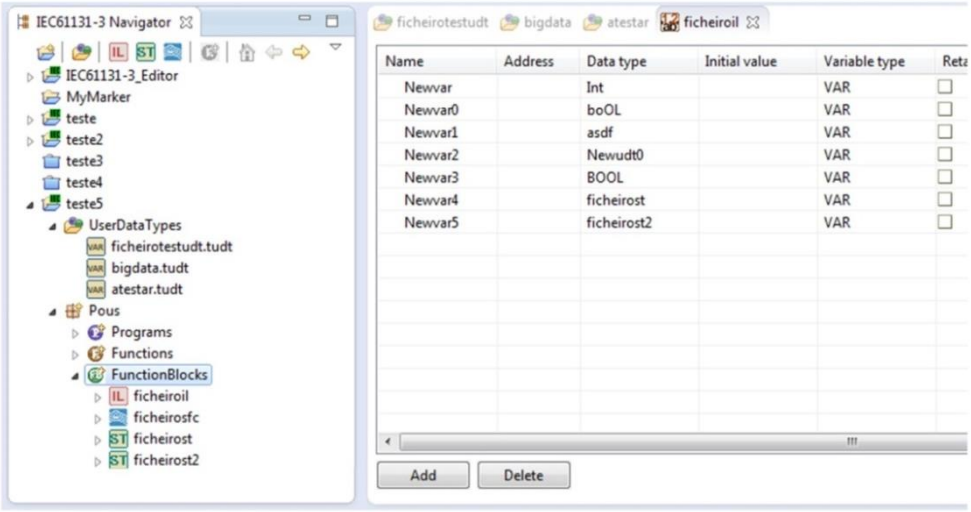

Figura 4.27 – Esquema para demonstrar o resultado da implementação de ***Observer patterns***. Com a criação do ficheiro a tabela é automaticamente atualizada (parte 3 de 3).

Na primeira parte é possível verificar que a variável "Newvar4" possui como tipo de dado o bloco funcional "ficheirost", o que faz com esta linha represente uma instância do bloco funcional, pelo que está validado. Porém, na variável "Newvar5" o tipo de dado não é validado, pois o valor textual, "ficheirost2" não é um tipo de dado. Pretende ser uma instância, mas o bloco funcional não existe.

Na segunda parte verifica-se a criação do bloco funcional com o nome "ficheirost2". Na terceira parte da figura verifica-se que o valor textual "ficheirost2" já foi validado. Esta validação ocorreu automaticamente sem intervenção do utilizador do programa, devido ao ser atualização a árvore de navegação enviou essa informação ao Editor de Variáveis que é seu subscritor.

Capítulo 5

Conclusões

Neste capítulo será apresentado um resumo dos resultados, apresentadas as conclusões do desenvolvimento e avaliado o trabalho desenvolvido. Serão também apresentadas propostas de trabalhos futuros para melhorar o desenvolvimento realizado e para a continuação do desenvolvimento do projeto onde se insere o desenvolvimento desta dissertação.

5.1 - Conclusões

Para a realização do desenvolvimento efetuado foi estudada em profundidade a Norma IEC 61131-3, nomeadamente as variáveis, os tipos de dados e a sua interação com os restantes elementos da norma. Foi também estudado o Eclipse e os seus projetos para a compreensão do funcionamento da plataforma, do desenvolvimento de *plug-ins* e também das ferramentas disponibilizadas. Além do estudo destas bases, para se poder realizar o desenvolvimento pretendido, foram também estudados os IDE concorrentes e existentes no mercado para uma melhor perceção do que o mercado procura e oferece, poder que o desenvolvimento realizado inove e acompanhe o que o mercado pretende. Por fim, foram estudados os desenvolvimentos já realizados no projeto.

Os objetivos propostos para esta dissertação eram quatro, tendo sido adicionado o objetivo número quatro no decorrer do desenvolvimento:

1) Implementação de um *plug-in* para a criação de variáveis, **Editor de Variáveis**.
2) Melhorar o *plug-in* que apresenta a árvore de navegação dos projetos da norma IEC61131-3, **Navegador**.
3) Implementação de um *plug-in* para a criação de novos tipos de dado, **Editor de UDT**.
4) Restruturação do projeto global e dos códigos dos *plug-ins* (objetivo acrescentado no decorrer da dissertação).
5) Implementação da ferramenta para fazer a importação de um projeto a partir de um ficheiro XML e a exportação de um projeto para ficheiro XML ou TXT.

Analisando de forma direta para os objetivos propostos foram atingidos 4 dos 5 objetivos propostos. O último objetivo, apesar de terem sidos projetadas e planeadas as ferramentas e de as classes terem sido preparadas para importar e exportar a informação, acabou por não haver tempo para ser realizada a implementação do desenvolvimento e para se proceder aos testes da mesma.

5.1.1 - Editor de Variáveis

O **Editor de Variáveis**, que a ideia era a criação de um simples editor livre, acabou por ficar completo que valida a informação dos nomes e dos tipos de dado, sendo que também permita a criação e validação de instância de blocos funcionais.

5.1.2 - Navegador

O **Navegador** foi melhorado e foram incorporadas uma série de funcionalidades que potencializam este *plug-in*, como por exemplo, a visualização de todos os projetos, a criação do **POUelement**, que agrupa os elementos do mesmo POU, a visualização e edição do tipo de retorno de uma função (function type) e a criação de POU de forma direta e mais simples. Ainda de salientar a criação de ficheiro do tipo de linguagem SFC de forma direta e a associação da extensão "*.sfc*" a edição dos diagramas gráficos.

5.1.3 - Editor de UDT

O trabalho realizado e as ferramentas criadas, no desenvolvimento do **Editor de UDT,** também lhe atribuem uma avaliação positiva, uma vez que foram criadas ferramentas úteis para *plug-ins* existentes, bem como para futuros *plug-ins*, como as pesquisas de padrões agrupadas na classe **Elementarchecks**, que foi aproveitada para validar também os nomes do **Editor de Variáveis**, pode ser usada nos atuais editores textuais e gráfico para a deteção de números, identificadores e valores literais, assim como poderá ser usado na criação dos restantes editores.

Ambos os editores foram desenvolvidos com o intuito de proporcionar ao utilizador uma agradável utilização que lhe proporcione uma fácil adaptação e que seja prático e rápido de trabalhar.

5.1.4 - Restruturação do código do projeto

Devido à restruturação do código e à centralização, no *plug-in* IEC61131_3, das classes que de informação genérica ao projeto confere ao projeto uma maior potencialidade para a partilha de informação entre as classes e permite uma maior independência entre os restantes *plug-ins*. A criação do elemento **TreeIECProject** permite uma melhor centralização da informação disponível no projeto para partilha da informação entre POU, os seus elementos (ficheiro de linguagem e variáveis) e UDT. A criação dos ***Observers*** permite uma melhor atualização da informação dos elementos do projeto.

Pode-se assim concluir que a maior parte dos objetivos foram atingidos com sucesso e que este foi um desenvolvimento que potencializa o projeto em curso.

5.2 - Trabalho Futuro

Nesta secção serão apresentados futuros desenvolvimentos quer ao trabalho desenvolvido quer ao projeto global.

5.2.1 - Futuros desenvolvimentos aos plug-ins criados

Relativamente aos editores, poderão ser realizadas melhorias ao nível da interface e partilha entre o editor e o sistema, como por exemplo integração com o *clipboard*, para permitir a ação de copiar e colar linhas, entre editores e por exemplo, entre programa de folha de cálculo, ou a possibilidade de importação e ou exportação de um ficheiro csv.

Outra melhoria realizada aos editores é criar ferramentas de validação dos valores na coluna relativa aos valores iniciais.

Um trabalho futuro de melhoria de integração dos editores com a árvore de navegação é a representação na árvore de navegação de ficheiros com erros (à semelhança do que acontece com o navegador de projeto do eclipse, quando uma classe tem um erro). Para a realização desta melhoria poderiam ser alterados os tipos de ficheiros "*tvar*" e "*tudt*", que além de guardarem as variáveis e UDT, respetivamente, poderiam passar a guardar informação de existência de valores inválidos. Esta informação estaria mais acessível ao **Navegador** para a representar.

A classe **TreeIECProject** pode ser desenvolvida para guardar mais informação do projeto, por exemplo, a criação de variáveis globais e variáveis de acesso. Também poderão ser criadas estruturas nessa classe para serem guardadrs as variáveis de entradas, saída e entrada e saída de POU para que os editores tenham acesso a essa informação.

Outras melhorias, mas de menor importância, serão aos menus de contexto, às *toolbars* e ações disponíveis, para a criação de novas entradas. Estas melhorias tanto podem ser realizadas nos editores, como na árvore de navegação (por exemplo, para renomear ficheiros).

5.2.2 - Futuros desenvolvimentos no projeto global

A integração das variáveis criadas nos editores textuais e no editor gráfico é um possível projeto futuro baseado nos plug-ins existentes.

Outro desenvolvimento futuro poderá ser a criação de Configuração, Recursos e Tarefas do projeto IEC 61131-3 que ainda não estão desenvolvidas.

Outros desenvolvimentos importantes que também foram referidos nas dissertações anteriores serão a integração do IDE com o compilador MATIEC e o desenvolvimento das linguagens gráficas em falta, a LD e a FBD.

Referências

[1] Karl Heinz John Michael e Tiegelkamp. *IEC61131-3: Programming Industrial Automation Systems*. Springer, 2001.

[2] *IEC 61131-3 Edition 2.0 (2003-01), TC/SC 65B, Programmable controllers - Part 3: Programming languages*

[3] *Eclipse Platform Overview*, Eclipse, Disponível em http://www.eclipse.org/eclipse/eclipse-charter.php Acesso em Outubro de 2015

[4] Filipe Ramos. *Eclipse IDE for the programming languages of the standard IEC 61131*. Dissertação de Mestrado, FEUP, 2014.

[5] José Ferreira. Ambiente integrado em Eclipse para desenvolvimento de programas IEC 61131-3. Dissertação de Mestrado, FEUP, 2015.

[6] *Introduction into IEC 61131-3 Programming Languages*, Disponível em http://www.plcopen.org/pages/tc1_standards/iec61131-3/index.htm Acesso em Novembro de 2015.

[7] *IEC 61131-3: a standard programming resource*. Disponível em http://www.plcopen.org/pages/tc1_standards/downloads/intro_iec.pdf Acesso em Novembro de 2015.